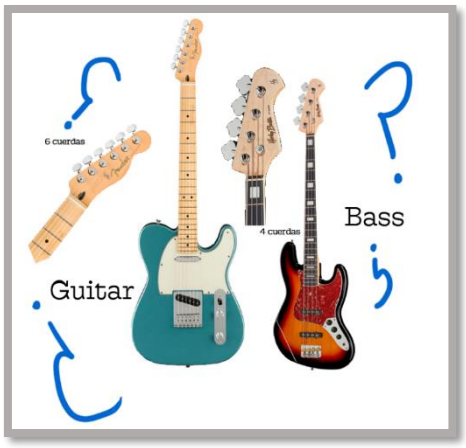
uc3m | Universidad Carlos III de Madrid

Programación Automática

Máster en Ciencia y Tecnología Informática



Trabajo Final de la Asignatura

Entrenamiento de un modelo de clasificación de bajos y guitarras a partir de un dataset de cosecha propia.

Alejandro Rey López

@alreylz

Contenido

1. Introducción

En este documento se detalla el proceso llevado a cabo para la creación de un modelo de clasificación capaz de discernir entre imágenes de guitarras y bajos a partir de un conjunto no preexistente, de recopilación propia del autor de este documento.

Para el entrenamiento de este modelo se ha realizado una recopilación de imágenes a través de *web scraping* [1], desarrollándose un programa para automatizar la descarga de las mismas desde una página web de venta de instrumentos. Dada la categorización de la propia página y el diseño minucioso de nuestra implementación, se logra obtener un total de 8000 imágenes perfectamente clasificadas sin esfuerzo humano adicional para la tarea de categorizado.

Asimismo, se describen los principales retos afrontados durante la etapa de pre-procesamiento de los datos, para hacerlos accesibles al modelo (pues no es posible introducir los datos al modelo de forma directa), la mayoría derivados de una escasa experiencia con el lenguaje de programación Python por parte del autor, así como por el tamaño considerable del *dataset* que se presenta en este trabajo.

Por último se presentan pruebas con diversas arquitecturas simples, de hasta 2 capas de convolución, con algunas mejoras potenciales como Batch Normalization [2] y o Dropout [3]. Se trata por tanto de, en la parte final del documento, explorar el espacio de modelos de forma limitada observando el comportamiento de la red y la calidad de inferencia que ofrece.

2. Descripción del problema

La guitarra y el bajo son instrumentos de cuerda con un parecido suficiente como para crear confusión entre las personas que no están familiarizadas con ninguno de los dos instrumentos. Tradicionalmente, la guitarra es un instrumento de 6 cuerdas con cuerdas más delgadas que las del bajo. El bajo, por su parte, suele ser más aparatoso y tener 4 cuerdas considerablemente gruesas.

Sin embargo, actualmente los factores diferenciadores entre ambos instrumentos se difuminan si se tiene en cuenta que existen bajos de 5, 6 e incluso más cuerdas; que los bajos y guitarras eléctricos presentan formas que se alejan de su aspecto tradicional, o que existen instrumentos de ambos tipos con formas muy similares entre sí.

Por este motivo, a menudo se encuentran noticias, o *rankings* de guitarristas en los que aparecen bajistas; eso sí, no al revés. Esto último se debe a la enorme popularidad de la guitarra, situada entre los instrumentos más ubicuos [4].

Motivado por la aparente dificultad del público general para distinguir estos dos instrumentos musicales, se plantea entrenar un modelo de clasificación en que permita categorizar imágenes de ambos instrumentos lo mejor posible. Se quiere por tanto realizar una prueba de concepto con una red neuronal para estudiar cómo de buenos son los resultados utilizando imágenes de baja resolución (*thumbnail*), de forma que se pueda estimar la aplicabilidad del uso de una red de este tipo para la clasificación de instrumentos usando imágenes de mayor tamaño.

3. Diseño de la solución

Para realizar el experimento se hubo de disponer de imágenes de los dos instrumentos en cuestión; la intención era poder proporcionar como entrada al modelo una gran variedad de guitarras y bajos, es decir, ejemplos de estos con distintas formas, diseños, clavijeros, etc. Puesto que estas son características que hacen difícil la generalización de lo que es una guitarra y lo que es un bajo respectivamente.

Por desgracia no fue posible encontrar un *dataset* apropiado que contuviese imágenes de ambos instrumentos, puesto que la guitarra, al ser más popular aparecía con mayor asiduidad en los conjuntos de datos explorados (como los de *ImageNet* [5]), pero no se pudo encontrar un *dataset* de tamaño comparable asociado a bajos.

A raíz de esta contingencia se decidió implementar un *web scraper* para la recopilación de imágenes que pudieran ser utilizadas como entrada para entrenar el modelo. En la **sección 3.1** se detalla brevemente el diseño del script de python que facilita esta la extracción de una gran cantidad de imágenes de ambos instrumentos y se proporciona acceso al mismo para que pueda ser consultado.

Una vez se completó la recopilación de imágenes, se procedió al pre-procesado de las mismas para poder introducirlas al modelo, además de generar las etiquetas o *labels* de clasificación, imprescindibles para el entrenamiento. Este proceso se detalla en la **sección 3.2.**

En la **sección 3.3**. se realizan pruebas de entrenamiento con el conjunto de datos recopilado, utilizando como base un Jupyter Notebook proporcionado para la práctica de *Pytorch* [6] de la asignatura de Programación Automática, accesible en [7]. Sobre este, se realizan las modificaciones oportunas para facilitar la reutilización de código y la flexibilidad a la hora de probar nuevas arquitecturas. En este apartado se detallan los resultados obtenidos tras una ligera búsqueda en el espacio de arquitecturas e hiperparámetros.

3.1. Web Scraping de Thomann

Para la recopilación de imágenes de forma masiva se desarrolló un script, que realiza peticiones y parsea páginas HTML, en este caso atravesando la estructura jerárquica de la página web Thomann [8] (una tienda online de instrumentos musicales) y buscando en este caso imágenes pertenecientes exclusivamente a guitarras y bajos para su descarga.

El código para la realización de este apartado está disponible como un *Jupyter Notebook*; y toda la ejecución se ha realizado a través de la plataforma *Google Colab* [9] , puesto que los recursos computacionales que posee el autor son demasiado limitados. Como se comentaba, el código que implementa lo que será detallado a continuación está accesible en [10].

El desarrollo utiliza como soporte las librerías *requests* [11] y *BeautifulSoup* [12]. Para la realización de peticiones y el *parsing* de tags HTML respectivamente.

A continuación, se documenta el diseño de este script a alto nivel, pues se puede consultar el Jupyter Notebook en caso de querer conocer el funcionamiento más a fondo:

3.1.1. Diseño de la solución

La página web *Thomann* tiene una estructura jerárquica donde los productos se agrupan por categorías (e.g. bajos/guitarras, drums, teclados), un diseño muy similar al de otras páginas de venta online, donde en la página HOME aparecen ofertas y novedades; hay un buscador y funcionalidades que permiten localizar productos por diversas categorías, tal y como evidencia la Ilustración 1.

Ilustración 1. Home de la web Thomann

Desde la página principal podemos observar las siguientes categorías de más alto nivel:

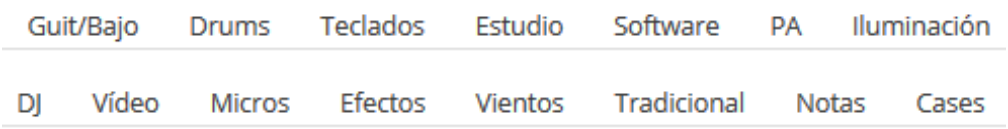


Ilustración 2. Subcategorías en página HOME de Thomann

El objetivo del web scraping es obtener imágenes de guitarras y bajos, por lo que el script desarrollado, en esencia, realiza una exploración recursiva de la jerarquía subyacente a la categoría “Guit/Bajo”.

Aunque la implementación parece trivial en un principio, en realidad, la categoría “Guit/Bajo” no solamente contiene productos de guitarras y bajos, sino que también aparecen instrumentos de cuerda con cierto parecido (Ukuleles, Banjos,etc.) y otros elementos asociados o no directamente a la guitarra y el bajo (como pedales, amplificadores, cuerdas, etc.)

Esto ocurre a su vez si vamos un paso más allá en la jerarquía de categorías, por ejemplo, si accedemos a la sub-categoría de Guitarras eléctricas (https://www.thomann.de/es/guitarras_electricas.html) donde vemos que nos aparecen *Sets de guitarra*, que incluyen varios objetos en los que no estamos interesados.

Estos conceptos de lo que se quiere preservar y lo que no , así como un ejemplo de jerarquía de categorías de instrumentos se muestra en en la Ilustración 3.

Dado que se desea entrenar una red con las imágenes que se obtengan de este *scraping* tratando de evitar un posterior procesado de imágenes manual o automatizado (lo que requeriría tiempo extra de trabajo para eliminar ruido del conjunto de entrenamiento), el adecuado filtrado de la exploración recursiva es necesario, además de la definición de un criterio de parada adecuado.

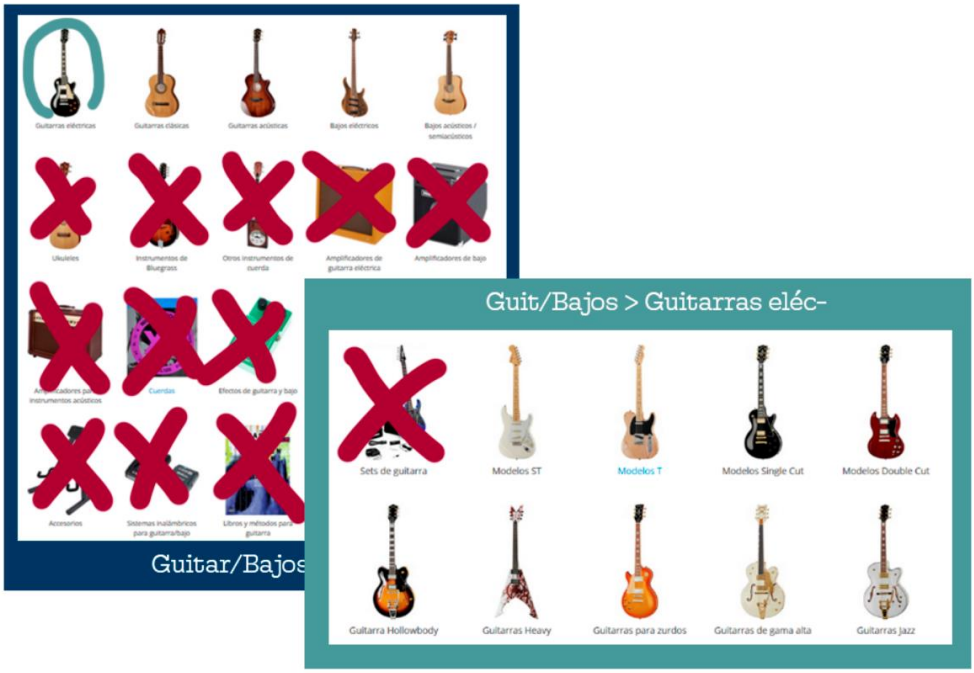


Ilustración 3. Jerarquía de categorías y elementos de interés.

Como se ha mencionado con anterioridad, todos los elementos que no sean guitarras o bajos no nos interesan y, por tanto, la mayor dificultad encontrada a lo largo del proceso de desarrollo ha sido la cuidadosa selección de filtros para descartar la exploración de ramas de hipertexto que no eran de interés.

Las imágenes finales se obtienen desde las *páginas de listado de productos* como puede ser la que se aprecia en la Ilustración 4. De estas se obtiene la url para la descarga de los recursos thumbnail, para luego realizar una petición y hacer persistente la imagen en Google Drive. Además, durante la exploración se guarda la *path* que se ha seguido para llegar a cada página de resultados, lo que facilitó la depuración y permitió el guardado de las imágenes en carpetas diferentes según la etiqueta (bajo o guitarra).

Además, como en cada página de resultados no es posible forzar que aparezcan más de 100 productos por página, se recorrieron todas las páginas de productos asociadas a cada página de resultados. Para conocer el número de páginas asociados a una página raíz, se exploró la página raíz en busca de los elementos HTML que permitían la paginación interactiva y se accedió al contenido del último enlace para determinar el número de páginas de resultados (e.g. 13, ver Ilustración 4).

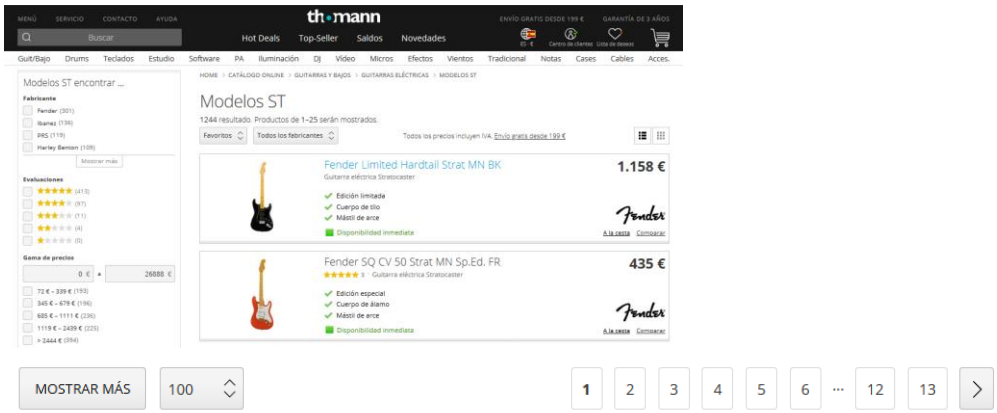


Ilustración 4. Ejemplo de página de resultados de productos y muestra de elementos de paginación (abajo), de los que se obtiene el número de peticiones a realizar para iterar sobre los resultados para cada tipología de guitarra o bajo.

El algoritmo simplificado que ha sido implementado para la creación del dataset propio aparece detallado en la Ilustración 5.

Pseudocódigo del algoritmo de *web scraping* para Thomann

Rellenamos una lista con pares: (path hasta categoría que lleva a enlace, url a página de resultados)

e.g. *("Bajos Eléctricos > Bajos J de 4 cuerdas", https://www.thomann.de/es/bajos_j_de_4_cuerdas.html)*

Llamaremos *resultPagesToExploreForDownload* a esta lista y una vez obtenida, exploraremos todas las url y las diferentes páginas de resultados asociadas para extraer una a una todas las imágenes encontradas.

Desde página de Guit/Bass: *https://www.thomann.de/es/guitarras_y_bajos.html*

For *enlace* in enlaces asociados a categorías que empiezan por "bajo o guitarra":

(a) *nuHTMLPage* ← Realizamos petición (request) al *enlace* (url) obtenido.

categoryURLList ← Exploramos *nuHTML* en busca de más sub categorías (más de HTML representan estas)

if *categoryName* no empieza por "Guitarra, Bajo o Modelo" no se añade la url y el nombre de la categoría a categoryURLList

// Se obtiene por tanto una lista de urls (*categoryURLList)* a otras subcategorías o a páginas de resultados que cumplen con los criterios anteriores. e.g. ("Bajos Eléctricos > Bajos J de 4 cuerdas" , "https://www.thomann.de/es/bajos_j_de_4_cuerdas.html"), ...

For *url* in *categoryURLList*

nuHTMLPage2 = Realizo request a *url*

if *nuHTMLPage2* es página de resultados then

añado a mi lista de páginas de resultados (*resultPagesToExploreForDownload*) con su etiqueta correspondiente, que se puede extraer de la path de exploración previa hasta llegar a cada página. e.g. (Bajo, url de página de resultados)

else: repetición de búsqueda en página de subcategorías vuelta a **(a)**

For *url* in *resultPagesToExploreForDownload*

nuHTMLpage ← Realizamos request(*url*) y parseamos response

//Obtenemos todas las url que corresponden a imágenes de artículos, además, atravesamos todas las páginas de resultados que aparezcan, si es que hay más de una

numPages = 1

if *(pagination)*: *numPages* ← getNumPages(*nuHTMLpage*)

for *pg* in *numPages*

thumbUrlList ← Obtenemos lista de urls de thumbnails de la página

For *urlTOfImange* in *thumbUrlList*:

Image = request(*urlTOfImange)*:

If *Image* not "empty" image.

Save(*Image, carpeta/etiqueta/ImageX.jpg)*

Ilustración 5. Pseudocódigo de alto nivel que describe el funcionamiento del programa de obtención de imágenes.

3.1.2. Resultados obtenidos

Tras este desarrollo y la ejecución del script enlazado anteriormente, se obtuvieron un total de 8000 imágenes. Se exploraron un total de 121 páginas de resultados, las cuales a su vez requerían la exploración de páginas de productos (puesto que no se permitía mostrar más de 100 productos por página). Entre estas páginas de resultados, al menos 29 eran páginas de bajos y 95 eran de guitarras.

De forma global, se obtuvieron 6924 imágenes de guitarras y 1076 de bajos de resolución 150x150 y modo RGB.

Las imágenes fueron guardadas en carpetas cuyo nombre está asociado a su etiqueta, de la forma que se observa en Ilustración 6.

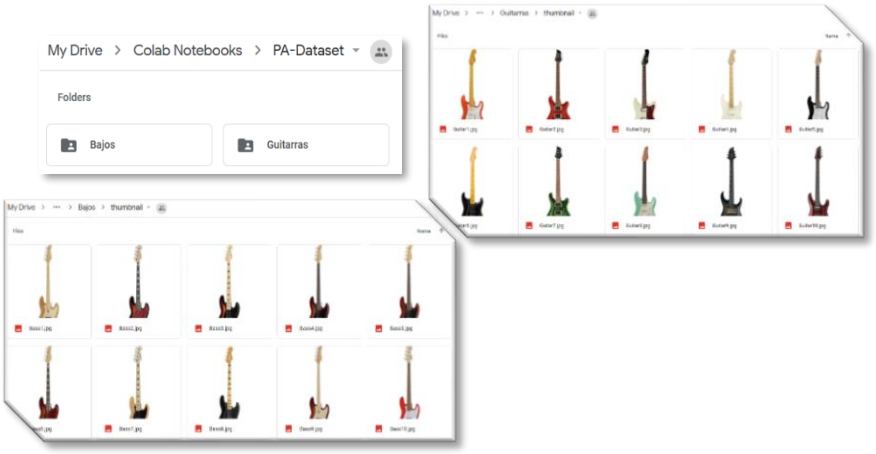


Ilustración 6. Esquema de guardado de las imágenes del dataset.

Todas las imágenes están accesibles en el *Drive* del autor de este documento [13].

3.2. Pre-procesado: imágenes como input del modelo de clasificación.

Una vez se logró la descarga masiva de imágenes en directorios tal y como se definió en la sección anterior, es necesario transformar los datos (imágenes y etiquetas) a un formato que un modelo implementado (en nuestro caso con PyTorch [6]) pueda comprender y recibir como entrada para entrenamiento e inferencia.

En esta etapa de pre-procesado, se llevan a cabo las siguientes tareas:

(1) Encapsulación de *dataset* en un archivo único.

Para evitar tener que cargar de disco la información de las imágenes e iterar en los directorios donde se encuentran las mismas cada vez que se quieren realizar pruebas sobre el modelo, se desarrolló un script para comprimir toda la información necesaria en un único archivo.

En un principio se realizó un proceso de conversión de cada imagen a una línea de un archivo CSV, dado que este era el formato en el que se codificó el dataset Semeion [14] para la práctica de PyTorch del comienzo de la asignatura. Sin embargo, esta estrategia junto con el uso de la librería *pandas* de Python para la carga y *pre-procesado* eran demasiado exigentes en términos de memoria (superando los 11GB de memoria alojados en Google Colab), lo que no permitía el progreso del proyecto y por lo cual se procedió a buscar una alternativa.

Finalmente se procedió iterar sobre los directorios de bajos y guitarras y transformar ambos elementos a *numpy arrays*, los cuales se pueden hacer persistentes a a través de la función *save()*, que resulta en un archivo con extensión *.npy* y parecen constituir una herramienta más escalable que la mencionada anteriormente.

Se obtienen por tanto del conjunto de datos global dos arrays que encapsulan todos los datos necesarios para el modelo:

- *guitarAndBassNpy*: numpy array de 4 dimensiones (Nº imagen, Height, Width, Channel)
- *labelsNpy*: numpy array de 1 dimensión que contiene un entero que representa la clase asocia da a cada imagen del array anterior (*guitarAndBassNpy*).

Estos archivos son guardados para evitar el proceso ineficiente de iterar sobre directorios que consume minutos de ejecución, mientras que la carga desde estos archivos ronda los 6 segundos de media.

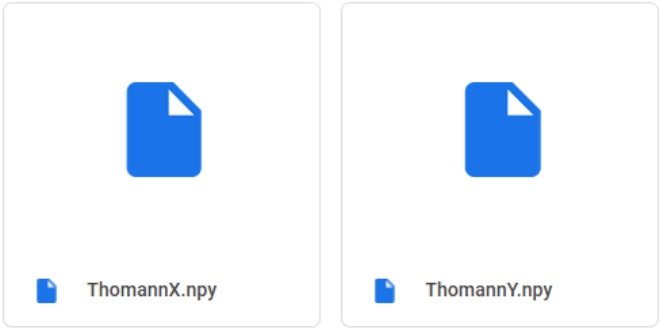


Ilustración 7. Dataset comprimido en 2 archivos, uno que contiene la lista de imágenes (ThomannX.py) y otra que guarda la etiqueta asociada a cada imagen (ThomannY.npy)

(2) *Shuffling* o barajado de imágenes y elementos etiquetas para balancear conjuntos de validación, test y entrenamiento.

Dado que al crear el array de etiquetas y el de imágenes primero se introdujeron los datos de guitarras y secuencialmente después los de bajos, la selección de subconjuntos de entrenamiento, validación y test compensados no es fácil de lograr.

Para solucionar este problema se procede a un barajado de las filas de los dos arrays *guitarAndBassNpy* y *labelsNpy* de forma simultánea para mantener su mapeo. Realizar esta operación es sencillo gracias a la función *shuffle()* de la librería *sklearn* , y si ejecutamos este proceso previo a hacer persistentes los conjuntos de datos, podemos ahorrar tiempo al cargar los datos ya prácticamente listos para introducirlos al modelo.

(3) División de dataset en subconjuntos para entrenamiento, validación y test

Para cada fase de las pruebas que se describen más adelante en el documento, se definen subconjuntos para las distintas fases del proceso, de los cuales, además, se comprueba el contenido (en bajos y guitarras), con el objetivo de verificar que el proceso de barajado ha sido útil.

La Ilustración 8 muestra el número y porcentaje de elementos del dataset dedicados a cada subconjunto del total de 8000 imágenes. Además, se muestran las dimensiones de cada array de datos y etiquetas y el contenido en guitarras y bajos tal y como se comentaba anteriormente

```
NUMBER OF ROWS in Dataset =  8000
        N_training (56%) = 4500 samples
        N_validation (18%) = 1500 samples
        N_testing (25%)  = 2000 samples
```

```
training: (4500, 150, 150, 3),(4500,)
validation: (1500, 150, 150, 3),(1500,)
testing: (2000, 150, 150, 3),(2000,)
Training set made of: 3880 guitars and 620 bass
Validation set made of: 1311 guitars and 189 bass
Testing set made of: 1733 guitars and 267 bass
```

Ilustración 8.Definición de subconjuntos de entrenamiento, validación y test; y su contenido.

(4) Permutado de dimensiones de los tensores para adecuar entrada al modelo.

El array de *numpy* que genera la función asarray() a partir de una imagen tiene 3 dimensiones, por lo que para incluir en la misma estructura una dimensión más es primeramente necesario utilizar la función *expand_dims()* de *numpy*. Pese a eso, el array resultante de la construcción por este medio tiene las dimensiones organizadas como sigue: *(Imagen,Width, Height, Channel)*. PyTorch, sin embargo, requiere que los tensores de entrada en imágenes RGB tengan el Canal RGB (Channel) como segunda dimensión, es decir, que sigan la *estructura (Imagen, Channel, Width, Height)*.

Para lograr esta conversión, utilizamos la función *permute()* de PyTorch, que permite la reordenación de las dimensiones. De esta forma, y con los parámetros que aparecen en la Ilustración 9.

```
torch.tensor(training_x,dtype=torch.float ).permute(0,3,1,2)
```

```
pre-tensor shape = (4500, 150, 150, 3)---> post-tensor shape =torch.Size([4500, 3, 150, 150])
```

Ilustración 9.Permutación de dimensiones para adecuar el tensor a lo que espera el modelo.

3.3. Pruebas y resultados

En este apartado se realizan una serie de pruebas con diversas arquitecturas de red, con hasta dos capas de convolución y una única capa lineal. Se realizan pequeñas variaciones en los hiper-parámetros con el objetivo de encontrar configuraciones que funcionen especialmente bien para este problema de clasificación. La Tabla 1. Resumen de resultados) resume los resultados obtenidos.

Para empezar, los periodos de entrenamiento son considerablemente largos, incluso ejecutando en GPU, rondando la media hora con tan solo una capa de convolución. Esto supuso un límite en cuanto a las pruebas que pudieron realizarse porque *Google Colab* impone unos límites de tiempo de ejecución en máquinas alojadas con GPU, es por esto que apenas se realizaron pruebas con 2 capas de convolución. Además, al dejar las pruebas más complejas para el final, algunas no se pudieron completar por los largos tiempos derivados de ejecución en CPU (ver ejemplo en Ilustración 10)

```
[Epoch: 10]

Time elapsed: 17.557307799657185
```

Ilustración 10. Tiempo de ejecución para 10 epochs en CPU alojada para 2 capas de convolución (extremadamente elevado)

A continuación, se describen brevemente los resultados obtenidos:

En primer lugar, es posible observar que el tamaño del *minibatch* influye en el tiempo de ejecución total del entrenamiento, disminuyendo la duración a medida que se incrementa su tamaño (esto se evidencia en T1, por ejemplo, que respecto a arquitecturas idénticas **[T2,T3]** obtiene el menor tiempo de todas las pruebas realizadas con un kernel de 5x5).

Por su parte, las primeras configuraciones probadas (**T1,T2,T3** en la tabla) , con una arquitectura de 1 capa convolucional , tamaño de kernel 5x5 y 5 mapas de características como salida; obtienen resultados mediocres y prácticamente equivalentes, independientemente de los parámetros que se utilicen a la entrada. Se observa en esta arquitectura poco más que diferencias en lo que se tarda en alcanzar el mejor *epoch* en validación, estancándose la precisión en test (Test Accuracy) en un **86.65%.**

En las siguientes pruebas (T5 y T6 en la tabla) , se decidió alterar el número de mapas de características para que la red tuviese mayor capacidad de abstracción, manteniendo el tamaño del kernel de la capa convolucional a 5x5. En **T5** con esta pequeña alteración de arquitectura, se observó una mejora hasta el 87.95%.

T6 añade la mejora de *batch normalization* a la arquitectura de T5 y esto tiene un efecto extremadamente positivo en los resultados, lográndose alcanzar valor de loss mínimo de 0.07 y una tasa de acierto en validación del **98.45%** , la segunda tasa en el global de pruebas realizadas.

T7 nace de la inclusión de una capa de *dropout* con probabilidad 0.5 en la arquitectura anteriormente descrita. Esta es la que mejores resultados obtiene, llegando al **99.25%** de acierto en validación. Con estas mejoras se evidencia que una red neuronal muy sencilla: 1 capa convolucional, *max-pooling*, *batch normalization*, *dropout* y una capa lineal; es capaz de resolver el problema de clasificación alcanzando una precisión muy alta. Cabe destacar también que es para el batchize más pequeño(32) y un momento de 0.3 y lr=0.003 (la mínima) que se logran estos resultados tan positivos.

El resto de arquitecturas varían el número de mapas de características de la capa 1, y a no ser que utilicen *batchnorm* no se acercan a los resultados anteriores (e.g. **T8** se queda en 95.3% en validación)

Para terminar, las única prueba completada para 2 capas convolucionales **(T10**)obtiene resultados muy alentadores (los segundos mejores globalmente), sin embargo, la mayor complejidad de la arquitectura y los elevados tiempos de entrenamiento hacen preferible una alternativa de una sola capa convolucional como es el caso de T6.

Los gráficos obtenidos para todas las arquitecturas y configuraciones mencionadas en este documento están accesibles en el Drive del autor [13].

Tabla 1. Resumen de resultados

	hyperparams			Performance						Conv 1				
ID	batchsize	lr	Momentum	Min Loss	Best epoch	Test Accuracy(%)	Time to best (min)	Total Training time(min)	Conv #	Kernel size	out maps	batch norm	DP	
T1	512	0.005	0.6	0.38	965	86.65	22.68	23.48	CONV 1	5x5	5	NO	NO	
T2	256	0.01	0.6	0.38	695	86.65	16.55	23.85	CONV 1	5x5	5	NO	NO	
T3	64	0.01	0.6	0.38	448	86.65	10.98	24.45	CONV 1	5x5	5	NO	NO	
T4	64	0.04	0.6	0.35	30	87.95	0.80	25.72	CONV 1	5x5	10	NO	NO	
T5	256	0.01	0.6	0.07	21	98.45	0.63	28.22	CONV 1	5x5	10	YES	NO	
T6	32	0.003	0.3	0.04	561	**99.25**	17.16	30.60	CONV 1	5x5	10	YES	0.5	
T7	256	0.01	0.3	0.06	52	98.75	1.67	29.68	CONV 1	3x3	20	YES	NO	
T8	512	0.005	0.3	0.24	19	95.3	0.44	21.55	CONV 1	3x3	5	NO	NO	
T9	512	0.005	0.3	NF	NF	NF	NF	NF	CONV2	5x5	10	NO	NO	(CPU EX
T10	512	0.01	0.6	0.05	215	98.86	9.60	**70.21**	CONV2	5x5	10	YES	NO	*30 out
	hyperparams			Performance										

3.4. Otros desarrollos y recursos

El script desarrollado incorpora el guardado automático del mejor modelo en validación para cada ejecución con una serie de parámetros. El conjunto de modelos pre-entrenados obtenidos de los experimentos realizados y mostrados en este documento, están accesible en la carpeta ***/Inference*** en [13]. Las tablas de resultados para cada modelo están también accesibles en una carpeta *sibling* **/ResultTables**.

Al final del script de la red neuronal [15] para clasificación, se ha implementado un sistema de carga de modelos para poder probar la red uno mismo seleccionando cualquier guitarra o bajo del *dataset*. Al ejecutar esta se nos presentan varios instrumentos y el usuario tiene que clasificarlos; posteriormente la red neuronal hace lo mismo y se puede observar de primera mano el funcionamiento correcto de la red en inferencia.

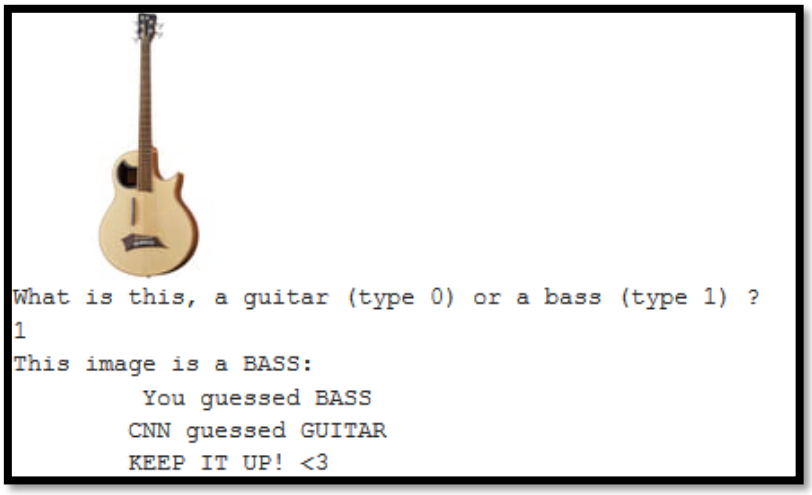


Ilustración 11. Ejemplo de pregunta del minujuego en el que te enfrentas a la red neuronal, este caso era para el modelo T4 (loss = 0.35)

4. Conclusiones y trabajo futuro.

En este trabajo se ha afrontado un problema de clasificación de imágenes de instrumentos musicales en con dos etiquetas posibles: bajo y guitarra. Para el entrenamiento de un modelo de red neuronal convolucional se recopilaron imágenes masivamente a fin de obtener imágenes de guitarras y bajos de todo tipo (eléctricos, acústicas, con diferentes formas, colores, etc.), aprovechando la base de datos de una página de venta online de instrumentos musicales (Thomann [8]). Se obtuvieron un total de 8000 imágenes y se hizo el *dataset* accesible libremente, así como el código desarrollado para realizar el scraping.

Se probaron multitud de configuraciones de red convolucional para la resolución de este problema, obteniéndose un porcentaje de clasificación del 99.25% en validación para una de las configuraciones, con una arquitectura muy sencilla.

Entre las **limitaciones** del trabajo destacamos:

En primer lugar, el problema descrito ha resultado ser muy sencillo para nuestro modelo, al existir solamente dos clases, un conjunto de datos amplio y dado que la probabilidad de acierto en clasificación es del 50% , se obtienen muy buenos resultados.

Por otro lado, las imágenes utilizadas eran de muy poca resolución y esto limita el uso del modelo de forma inmediata en imágenes que un usuario final manejaría en su vida cotidiana. Las arquitecturas utilizadas ofrecen buenos resultados, pero habría que estudiar si al añadir clases, los resultados se mantendrían. Además, todas las fotografías extraídas tenían fondo blanco y esto podría tener algún efecto negativo en la generalización, pues podríamos decir que se ha entrenado con imágenes que muestran el instrumento principalmente de frente, ligeramente girado o reposando en el suelo.

Sería interesante de cara al futuro probar de primera mano la calidad de la inferencia con imágenes de mayor tamaño tras aplicar *downscaling*, de forma que pudiésemos observar si es posible adaptar de forma inmediata el modelo para reconocimiento de imágenes de instrumentos de mayor resolución a costa de un tiempo menor de entrenamiento.

En referencia a la calidad del dataset recopilado, se han detectado algunos elementos que la perjudican:

(1) Se detectaron imágenes de *sets de guitarras* o bajos incluso en esta categoría, dado que al ser un "producto", a menudo se vende la guitarra como artículo principal pero el modelo concreto va acompañado de púas o una funda y por tanto aparecen estos complementos en la imagen *thumbnail*. A partir de la exploración manual del conjunto de guitarras, estimamos que este tipo de imágenes son muy pocas en comparación con el total de imágenes y por tanto esto no compromete de ninguna forma los resultados obtenidos, permitiendo detectar guitarras junto a una funda.
(2) Existe una descompensación entre los ejemplos de guitarras y bajos que se proporcionan al modelo. De cara al futuro se podría realizar un desarrollo de otro *scraper* para mitigar este problema.

Como es natural en las redes neuronales artificiales, es difícil interpretar lo que hace que el modelo clasifique correctamente los instrumentos, es decir, conocer qué características lo hacen distinguir ambos. Sería interesante explorar cuáles son las características que es capaz de abstraer el modelo con métodos como los propuestos en [16].

Bibliografía

[1] Wikipedia, the free encyclopedia, «Web Scraping,» [En línea]. Available: https://en.wikipedia.org/wiki/Web_scraping. [Último acceso: 28 Mayo 2020].

[2] S. y. S. C. Ioffe, «Batch normalization: Accelerating deep network training by reducing internal covariate shift,» de *32nd International Conference on Machine Learning, ICML 2015*, 2015.

[3] N. Srivastava, G. Hinton, A. Krizhevsky y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting,» *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[4] ABRSM, «Associated Board of the Royal Schools of Music: A Social and Cultural History,» [En línea]. Available: https://gb.abrsm.org/en/making-music/4-the-statistics/42-shifts-in-instrumental-trends/. [Último acceso: 28 Mayo 2020].

[5] Stanford Vision Lab, Stanford University, Princeton University, «IMAGENET,» 2016. [En línea]. Available: http://www.image-net.org/index.

[6] PyTorch, «PyTorch,» [En línea]. Available: https://pytorch.org/. [Último acceso: 28 Mayo 2020].

[7] Programación Automática, Máster en Ciencia y Tecnología Informática, «semeion_conv.ipynb,» [En línea]. Available: https://colab.research.google.com/drive/1LADm62h5JNwuUrEtWez-1Pjg1cxMtrDm#scrollTo=xZdcCqv_MlQ9.

[8] Thomann GmbH , «Bienvenidos - Thomann España,» [En línea]. Available: https://www.thomann.de/es/index.html. [Último acceso: 28 05 2020].

[9] Google, «Te damos la bienvenida a Colaboratory - Colaboratory,» [En línea]. Available: https://colab.research.google.com/. [Último acceso: 28 Mayo 2020].

[10] A. R. L. @coredamnwork, «Thomann Web Scraping,» Google Colab, 2020. [En línea]. Available: https://colab.research.google.com/drive/1WRxX31jkLKDV7QFKwbm5Wsa_W0uYUPv9.

[11] K. Reitz, «Requests: HTTP for Humans™,» [En línea]. Available: https://requests.readthedocs.io/en/master/. [Último acceso: 28 Mayo 2020].

[12] L. Richardson, «Beautiful Soup 4.9.0 documentation,» [En línea]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Último acceso: 28 Mayo 2020].

[13] A. R. L. @coredamnwork, «Google Drive | Dataset BassAndGuitar CNN,» [En línea]. Available: https://drive.google.com/drive/u/0/folders/12QiXJj9cZJzijI63JD5-70n2o_QhvOj5.

[14] Semeion Research Center of Sciences of Communication, «Semeion Handwritten Digit Data Set,» [En línea]. Available: https://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit. [Último acceso: 28 Mayo 2020].

[15] A. R. L. @coredamnwork, «Bass&Guiutar Convolutional Net @coredamnworl.ipynb,» [En línea]. Available: https://colab.research.google.com/drive/10eAJew8CF3LHK38K-FYbBlUIVZGknLEQ. [Último acceso: 29 Mayo 2020].

[16] Stanford CS, «CS231n Convolutional Neural Networks for Visual Recognition,» [En línea]. Available: https://cs231n.github.io/understanding-cnn/.

[17] Alejandro Rey López @coredamnwork, «Google Drive - PA-Dataset,» [En línea]. Available: https://drive.google.com/drive/u/0/folders/12QiXJj9cZJzijI63JD5-70n2o_QhvOj5. [Último acceso: 29 Mayo 2020].