Reconocimiento de partituras musicales

Toni Santaella Trujillo

Resumen– El Optical Music Recognition (OMR) es un campo de investigación que investiga como leer computacionalmente notación musical en documentos, dicho campo ha demostrado ser díficil de tratar debido a la calidad de los resultados que se obtiene. Muchos han intentado acercarse a este problema usando métodos de Deep Learning, diviendo el problema en diferentes pasos como el reconocimiento de los símbolos musicales de la partitura y la reconstrucción de la información musical para poder formar un modelo que de una descripción de esta partitura. La mayoría de sistemas OMR solo cubren una parte de todo este sistema sin llegar a producir un archivo con el que podamos escuchar la notación musical que estos analizan. En este documento se propone un modelo *sequence-to-sequence* capaz de producir archivos MIDI a partir de imágenes que contengan notación musical.

Palabras clave – seq2seq, WAV, OMR

Abstract– Optical Music Recognition (OMR) is research field that has proven to be difficult to deal with because of the quality of the results researchers have obtained. Many have tried to approach this problem using Deep Learning methods, breaking the problem into different steps as music symbol recognition and musical information reconstruction to form a model that can give a description of the sheet. The majority of this OMR systems only cover a part of all this system without getting to produce a file which can reproduce the musical notation they are analyzing. In this paper we propose a sequence-to-sequence model that can produce MIDI files from images containing musical notation.

Keywords– seq2seq, WAV, OMR

1 Introducción - Contexto del trabajo

Optical Music Recognition (OMR) es el campo de investigación que investiga como leer notaciones musicales en documentos [1]. Este trabajo se acercará de una manera peculiar al problema del OMR, en vez de detectar y reconocer la partitura en un primer lugar para luego poder generar el archivo de audio, nos saltaremos la parte de reconocer y directamente generaremos este archivo de audio que puede ser formato WAV o un archivo MIDI.

La motivación principal de este trabajo viene de poder ayudar a gente que quiera aprender música. El resultado final permitirá que a partir de la imagen de una partitura musical podamos obtener un archivo de audio el cual contenga el sonido de la partitura.

En este trabajo me voy a centrar en la mejora de un modelo secuencia-secuencia [2], al cual a partir de este momento me referiré como seq2seq o simplemente modelo. Los modelos seq2seq se basan en una arquitecura codificador-decodificador, están siendo muy utilizados últimamente ya que permiten que los datos de entrada y salida tengan distinto tamaño y modalidad, esto hace posible que podamos introducir una imagen como dato de entrada y obtener algo totalmente distinto como un archivo de audio en la salida. Para desarrollar esta mejora se va a crear una variante del modelo la cual trabaje sobre pentagramas en vez de compases. Al final del documento podremos ver una comparación de esta variante con el original. Analizaremos los resultados y los procesos que son eliminados gracias a trabajar con pentagramas directamente. Entraré en más detalle sobre este tipo de modelos y su arquitectura a lo largo del documento.

El modelo seq2seq que utilizaré entrena con unas bases de datos de partituras sintéticas [3], recibe una partitura como input y nos devuelve como output un archivo MIDI con la partitura en sonido.

1.1 Objetivos

El objetivo principal es crear un programa que nos permita a partir de imágenes de partituras musicales extraer la información de estas y crear un archivo de sonido el cual

• E-mail de contacte: tonisantaella20122000@gmail.com
• Mención realitzada: Computació
• Treball tutoritzat per: Alicia Fornes Bisquerra (Ciencias de la Computación)
• Curs 2021/22

contenga la reproducción de las mismas.

Aunque el objetivo principal parezca bastante directo, es necesario su división en varios subobjetivos, ya que el problema del OMR abarca varias tareas a la vez, tales como la detección en imágenes, predicción del sonido o generación de partituras. Los subobjetivos son los siguientes:

- **Estudiar estado del arte**: en los últimos años se han publicado menos de 500 trabajos individuales sobre OMR, la poca consideración entre estos y la falta de uso de trabajo previamente creado por otros investigadores llevan a los nuevos investigadores a invertir tiempo en un trabajo que probablemente ya haya sido realizado antes [1].
- **Estudiar arquitectura seq2seq**: en este trabajo se trabaja sobre un modelo seq2seq, por lo tanto es necesario el conocimiento sobre esta arquitectura para poder alcanzar el objetivo principal.

Para la realización de todo lo explicado anteriormente, partimos de lo siguiente:

- **Generador automático de partituras**: script de generación de partituras sintéticas para entrenar el modelo de inteligencia artificial. Es capaz de generar partituras sintéticas con el grado de dificultad que queramos (distorsión para simular partituras escritas, notas simples, sin pentagrama, etc).
- **Modelo secuencia-secuencia**: modelo seq2seq que vamos a usar para entrenar. Este modelo genera archivos de texto los cuales contienen la transcripción musical. Esta característica del modelo se cambiará para que podamos generar archivos MIDI los cuales podremos usar para escuchar las notaciones musicales.

1.2 Metodología

Todo el trabajo se realizará siguiendo la metodología Kanban. Con esta metodología se pretende tener una visualización del flujo del trabajo constante controlando las tareas a realizar y organizar las tareas de manera eficiente. Se dividirán las tareas en tres columnas donde se podrán ver las tareas que hay que realizar, las que se están realizando y las que han sido completadas. Usaré la herramienta Trello [4].

1.3 Planificación

Para la correcta planificación del trabajo es necesario la división de este en tareas. Ya se ha definido antes los objetivos de este y hasta donde se planea llegar, a continuación podemos ver las tareas que habrá que realizar para poder completar los objetivos propuestos.

1. Estudio del estado del arte.
   (a) Documentación sobre OMR.
   (b) Documentación sobre modelo seq2seq y su arquitectura.
2. Puesta en marcha de la línea base.
   (a) Entender el algoritmo que genera las partituras.
   (b) Generar bases de datos.
   (c) Usar el modelo seq2seq.
3. Modificación del modelo base seq2seq para que este pueda generar archivos MIDI.
4. Obtención de resultados.
5. Documentación del trabajo.

2 Estado del arte

En la actualidad existen diferentes aplicaciones que tratan el tema del OMR, aunque la mayoría de información sobre este campo se encuentra en artículos científicos e investigaciones.

A nivel software podemos encontrar diferentes aplicaciones como PlayScore 2, ScanScore, o Audiveris [5]. Esta última es una opción de código abierto. La gran mayoría de las aplicaciones funcionan siguiendo un pipeline parecido al siguiente [6]:

1. Preprocesado de la imagen.
2. Reconocimiento de las figuras musicales.
3. Reconstrucción de la información musical.
4. Construcción del modelo de notación musical

En relación a los artículos científicos, algunos hacen uso de CNNs, como por ejemplo el siguiente trabajo [7] que demuestra una mejora en la detección de las líneas de un compás al usar CNNs.

Las figuras musicales son clasificadas en base a su forma y las similitudes que presentan. Para su correcta clasificación se ha llegado a usar *template matching*, máquinas de vectores de soporte, redes neuronales, etc. Debido a la popularidad de las redes neuronales profundas (DNN), estás se están usando mucho ahora para esta tarea.

La razón por la que la mayoría de sistemas OMR es suelen involucrar una cantidad considerable de pasos, como por ejemplo la identificación de pentagramas o la clasificación de objetos. Cada uno de estos pasos tiene sus dificultades y esto hace que se acumulen muchos errores a lo largo del proceso. El artículo presentado por Rebelo [8], contiene información sobre los sistemas OMR y su estado del arte, en el se describe que estos sistemas OMR ”comunes”, después de realizar el pre procesado de la imagen (reducción del sonido, binarización, etc) pueden dividirse en tres módulos principales:

1. Reconocimiento de los símbolos musicales en la partitura.
2. Reconstrucción de la información musical para construir una descripción lógica de la notación musical.
3. Construcción de un modelo de notación musical para la generación de la descripción simbólica de la partitura.

En este artículo [9] se tiene en cuenta lo problemático que es todo esto y se opta por reducir todos estos problemas con el uso de redes convolucionales y recurrentes, resultando en un modelo convolucional *sequence-to-sequence*.

3 LÍNEA BASE

3.1 Algoritmo generador de partituras

Los datos para entrenar el modelo seq2seq se obtienen de un algoritmo que genera partituras aleatorias. Este algoritmo nos genera las partituras a nivel de compás. El algoritmo hace uso de la librería Lilypond [10] que es la encargada de generar las imagenes PNG y su groundtruth.

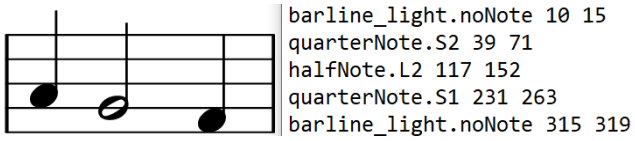

Fig. 1: Ejemplo de un compás sencillo con dos notas negras y una blanca

Por cada partitura que se crea el algoritmo genera entre unos 100 - 120 compases. Podemos elegir los diferentes componentes que pueden estar presentes en la partitura, es decir, podemos decidir el grado de dificultad de esta. Esto quiere decir que podemos tener una partitura con figuras sencillas como notas negras y blancas (Fig. 1), o ir un poco más allá y hacer que la partitura pueda tener acordes, legatos, ligaduras, ... (Fig. 2).

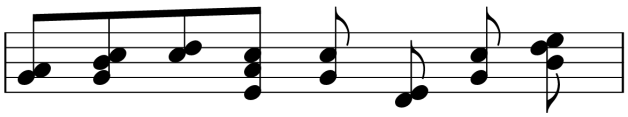

Fig. 2: Partitura más compleja

Evidentemente, a más dificultad tenga la partitura más difícil le será al modelo identificar correctamente todas las figuras musicales.

3.2 Dataset

Con el algoritmo explicado en el punto anterior podemos crear diferentes datasets con diferentes características, con partituras más sencillas o complejas dependiendo del objetivo. En el dataset usado para entrenar el modelo que tenemos en la línea base podemos encontrar partituras simples con notas muy básicas (blancas, negras, corcheas, ...).

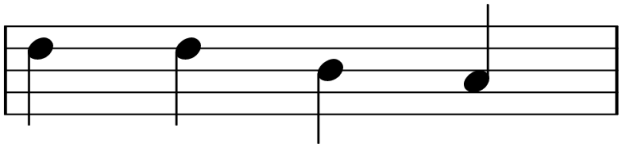

Fig. 3: Partitura presente en el dataset

Cada imagen es preprocesada de la siguiente manera:

1. Normalización.
2. Redimensión de la imagen a la altura y anchura deseada.
3. *Image Augmentation* (si activado): se amplía el dataset aplicando *augmentation* con el fin de aumentar el número de imágenes de entramiento, haciendo más robusto el sistema ante deformaciones. En el caso de estar activado se realizan las siguientes modificaciones:
   (a) Salt&Pepper.
   (b) Desenfoque de lente o *image sharpening*
   (c) Correción gamma.
4. Se aumenta el número de canales a 3 y se aplica *VGG normalization*.

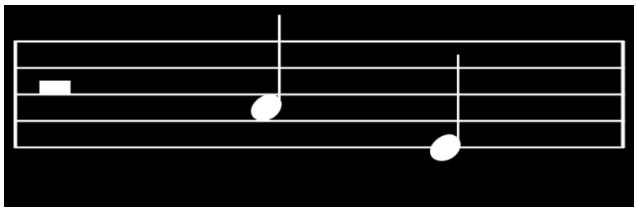

Fig. 4: Imagen después de pasar por el paso 2

3.3 Modelo Seq2Seq

Un modelo seq2seq está basado en redes neuronales recurrentes (RNN). Este tipo de redes surgen de la motivación de recordar la información que van aprendiendo. Son efectivas contra secuencias largas de datos lo que las hacen más eficientes en problemas como la traducción de texto.

Este tipo de redes funcionan con celdas LSTM las cuales son capaces de añadir o borrar información del estado de la celda mediante una serie de compuertas. Estas compuertas son redes neuronales que pueden dejar pasar la información o bloquearla. En total, una celda LSTM tiene 3 compuertas: *forget gate*, *input gate*, *output gate*.

Dicho esto, podemos optar por usar GRUs en vez de LSTMs. Las GRUs son una variación de las LSTMs las cuales están siendo muy populares últimamente. Es un modelo más simple que combina las dos primeras compuertas en una: *update gate*.

Finalmente, teniendo en mente todo esto sobre las celdas GRU y LSTM [11] podemos indagar más en el modelo seq2seq. Este modelo lo podemos llamar de otra manera como un modelo encoder-decoder. El encoder procesa toda la información en el dato de entrada y genera un vector. El decoder recibe el vector que genera el encoder e intenta predecir el target, en nuestro caso intentará predecir el sonido del compás.

La siguiente estructura que voy a explicar está relacionada con los mejores resultados que se han obtenido con el modelo usado:

- Encoder: los datos en el encoder pasan primero por una red convolucional profunda (VGG19). Esta red consta de hasta unas 19 capas de convolución. La salida, si está activa la opción de dropout, pasará también por una capa de dropout. Finalmente pasa por una RNN formada por celdas GRU.
- Decoder: el decoder funciona con un mecanismo de ”atención”. Este mecanismo permite al decoder enfocarse en partes importantes antes de generar el output,

esto permite obtener unos mejores resultados. El mecanismo de atención que utiliza el modelo es una combinación del mecanismo de atención estándar Bahdanau y atención de ubicación.

En la siguiente figura podemos ver de forma gráfica y simplificada la arquitectura del modelo seq2seq

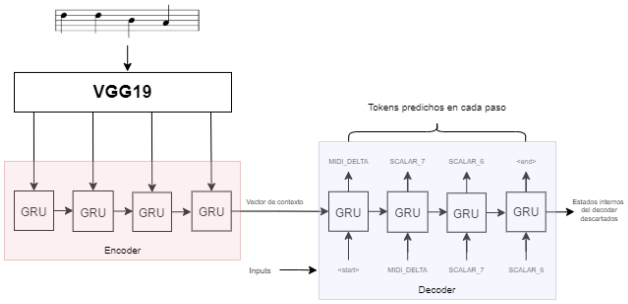

Fig. 5: Arquitectura del modelo seq2seq

3.4 Entrenamiento

El proceso que sigue el modelo seq2seq una vez ha cargado todos los datos del dataset que se le ha proporcionado lo podemos dividir en dos fases.

En la primera fase de entrenamiento preparamos el encoder y el decoder con datos para actualizar los pesos de nuestra red. El encoder es alimentado con los datos de entrada y produce el vector de contexto, este se pasa al decoder y produce una salida, así para cada imagen del dataset. La segunda fase es de inferencia, en esta fase probamos el modelo con datos que no ha visto nunca antes.

Los resultados obtenidos se evalúan con el SER (Symbol Error Rate) y una función de pérdida que puede ser *cross_entropy* [12] o *KLD Loss* (Kullback-Leibler divergence loss) [13] si está activado el *Label Smoothing*.

La primera función de pérdida calcula como su nombre indica, la pérdida de entropía cruzada entre el input y el target. Resulta útil cuando estamos en un problema de clasificación con diferentes clases y con un dataset de entrenamiento que no está balanceado.

La segunda función de pérdida calcula la pérdida de la divergencia de Kullback-Leibler (aunque también se le puede llamar entropía relativa). Este calculo se realiza entre dos tensores del mismo tamaño, el primer tensor tiene el tamaño del input y el segundo el del target. La pérdida resultante es la diferencia entre el primer tensor y el segundo.

Entrenando el modelo que tenemos hasta ahora con una serie de parámetros podemos obtener los siguientes resultados:

TABLA 1: RESULTADOS

Loss	0.074
SER	0.0007%

Podemos observar que la función de pérdida da un valor bastante bajo, por lo que no suele haber diferencias apenas entre el target y lo que predice el modelo. El SER también da un valor bajo, esto quiere decir que la cantidad de elementos que ha reconocido es bastante alta, siendo muy poco probable que añada notas de más en el audio o menos.

4 DESARROLLO DEL TRABAJO

4.1 Pre procesado

Una de las primeras mejoras a tener en cuenta es hacer que el modelo funcione con partituras enteras. Hasta ahora el modelo era alimentado con compases y nos devolvía un archivo con los tokens correspondientes los cuales hay que transformar a MIDI, aunque esta parte se explicará con más detalle más adelante. Nosotros queremos que quien use el modelo pueda hacer una foto a una partitura entera, o un pentagrama, y el modelo extraiga los compases de esta.

El proceso que se sigue es el siguiente:

1. Recorte de la partitura.
2. Recorte de los pentagramas.
3. Recorte de los compases.

A continuación vamos a mostrar ejemplos de este proceso, el pentagrama en cuestión el cual va a ser procesado es el de la siguiente figura.

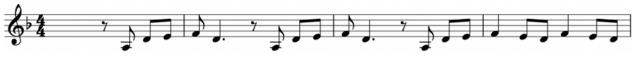

Fig. 6: Pentagrama usado para recortar los compases

La foto de la partitura no tiene porque ser exactamente cuadriculada, por ello necesitamos encontrar el contorno más grande en la imagen, que en este caso sería la hoja donde está impresa la partitura. Con el contorno más grande se aplica una *four point transformation* por si la imagen no está del todo recta.

Para los recortes de los pentagramas realizamos una proyección horizontal para encontrar en que regiones de la imagen, horizontalmente, hay píxeles dibujados. De esta manera podemos encontrar donde empieza y termina cada uno de los pentagramas y poder recortarlos. Encontrar estas regiones es más sencillo en comparación al siguiente paso. Esto se debe a que normalmente las zonas donde empiezan a haber píxeles suelen ser el comienzo de un pentagrama, y cuando vuelve a haber 0 píxeles es porque el pentagrama ha terminado.

Finalmente para los compases realizamos operaciones de morfología para eliminar las líneas horizontales del pentagrama ya que solo nos interesan las verticales que son las líneas divisorias. Esta parte del pre procesado es la que se complica más, como podemos observar en la figura 7, tenemos varias regiones. Las líneas divisorias normalmente son las más estables, esto quiere decir que se muestran casi constantes en el valor de píxeles. Con esto en mente podemos pensar en un método para poder aislar estas zonas y

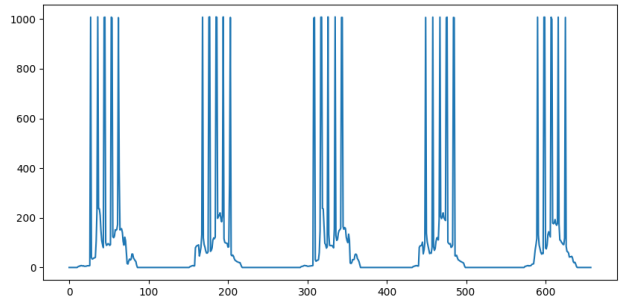

Fig. 7: Proyección horizontal de una partitura

recortar correctamente los compases. Por ejemplo, sabemos que si la línea divisoria se mantiene casi siempre constante en el valor sus píxeles tendrá una varianza baja en comparación a las otras figuras musicales que haya en el pentagrama. Sabemos también que las líneas divisorias ocupan el mismo espacio siempre (la altura del pentagrama), por lo que si encontramos figuras musicales que tengan poca varianza (como un silencio) las podemos descartar ya que estas no ocupan tanto espacio en el pentagrama como las líneas divisorias.

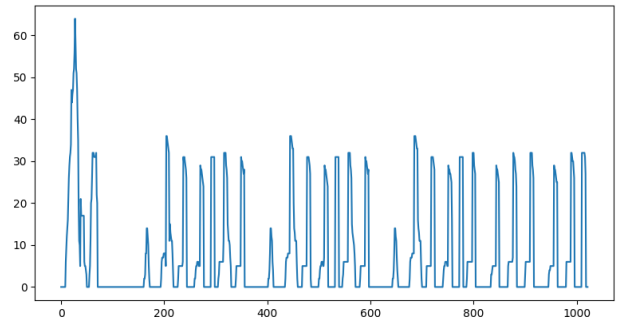

Fig. 8: Proyección vertical de un pentagrama

4.2 Post procesado

El modelo en la salida nos genera un archivo con los tokens de las partituras, estos tokens son solo abstracciones de los valores MIDI de *time* y *pitch*. En la figura 8 podemos ver lo que el modelo extrae de un compás cualquiera con el que ha sido alimentado. La parte de post procesado se encarga de convertir estos valores en los valores de tono y tiempo y crear un archivo MIDI a partir de ellos.

El archivo MIDI está organizado en diferentes pistas (*tracks*). Cada pista contiene información sobre el valor del tono, tiempo y velocidad de la nota. Juntando estas pistas podemos componer el sonido del compás, del pentagrama, y finalmente la partitura entera.

```
MIDI_DELTA SCALAR_7 SCALAR_6 SCALAR_8 MIDI_NOTEON MIDI_PITCH SCALAR_6 SCALAR_5 MIDI_DELTA SCALAR_1 SCALAR_9 SCALAR_2 MIDI_NOTEOFF
MIDI_PITCH SCALAR_6 SCALAR_5 MIDI_DELTA SCALAR_0 MIDI_NOTEON MIDI_PITCH SCALAR_6 SCALAR_5 MIDI_DELTA SCALAR_1 SCALAR_9 SCALAR_2
MIDI_NOTEOFF MIDI_PITCH SCALAR_6 SCALAR_5 MIDI_DELTA SCALAR_0 MIDI_NOTEON MIDI_PITCH SCALAR_6 SCALAR_5 MIDI_DELTA SCALAR_1
SCALAR_9 SCALAR_2 MIDI_NOTEOFF MIDI_PITCH SCALAR_6 SCALAR_5 MIDI_DELTA SCALAR_0 MIDI_NOTEON MIDI_PITCH SCALAR_6 SCALAR_5
MIDI_DELTA SCALAR_1 SCALAR_9 SCALAR_2 MIDI_NOTEOFF MIDI_PITCH SCALAR_6 SCALAR_5
```

Fig. 9: Tokens

El significado de los tokens es el siguiente:

- MIDI_DELTA: los siguientes valores SCALAR_X forman el *time* de la nota.
- MIDI_NOTEON: la nota tiene un valor de velocidad 90.
- MIDI_NOTEOFF: la nota tiene un valor de velocidad 0.
- MIDI_PITCH: los siguientes valores SCALAR_X forman el *pitch* de la nota.
- SCALAR_X: es un número del valor *time* o *pitch*, si los valores son SCALAR_7 y SCALAR_8, el número será 78.

4.3 Dataset de pentagramas

Recortar los compases es un proceso complicado, depende mucho de que la imagen esté bien realizada y puede haber errores si la calidad es mala. Pero por otra parte, recortar los pentagramas es mucho más sencillo. Es por esto que hemos decidido entrenar el modelo, ahora, con pentagramas enteros en lugar de compáses y comparar así los resultados para ver que opción sería mejor.

El dataset consta de aproximadamente 25 mil imágenes de pentagramas y sus correspondientes archivos .MIDI.

Fig. 10: Pentagrama corto del nuevo dataset

Estos pentagramas tienen figuras musicales sencillas como el dataset de compases, además de que pueden tener un compás o varios, por lo que el dataset es una mezcla de compases y pentagramas.

5 Resultados

5.1 División de compases

Como se ha comentado en el punto anterior, la división de compases es un proceso complicado y puede haber errores. Tomemos en cuenta la siguiente imagen tomada desde un teléfono móvil de una partitura sencilla (figura 11). La partitura tiene 5 pentagramas y cada uno tiene un distinto número de compases pero solo vamos a fijarnos en los resultados del primer pentagrama y del segundo.

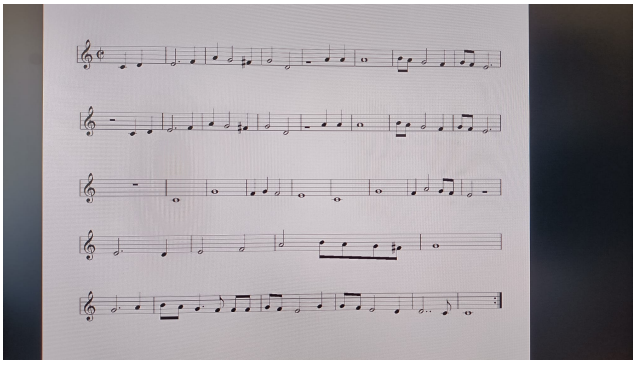

Fig. 11: Foto tomada de una partitura

En las figuras 12 y 13 podemos observar que los pentagramas son recortados correctamente, en esta parte no hay problemas ya que con la proyección horizontal es sencillo encontrar donde están los pentagramas.

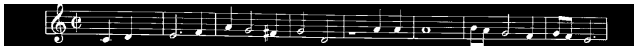

Fig. 12: Primer pentagrama

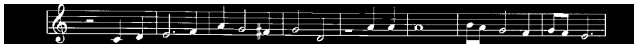

Fig. 13: Segundo pentagrama

A la hora de recortar los compases podemos tener distintos problemas. Estos problemas son los siguientes:

- Tras el proceso de binarización de la imagen es posible que haya ruido. Debido a esto las líneas divisorias las cuales tendrían que mantenerse constantes en todos los valores de sus píxeles podrían verse alteradas, tener un valor dispar a todos los demás y provocar que la varianza suba. Normalmente esto no supondría un problema pero hay que tener en cuenta que las líneas divisorias ocupan poco espacio horizontalmente y esto hace que tengamos pocos valores para realizar el cálculo de la varianza. Esto quiere decir que tener un valor distinto al resto por pocas unidades haría que la varianza subiera mucho.

- Al aplicar morfología es posible que las notas separen la cabeza de su cuerpo. Esto supondría que el cuerpo se detectaría como una figura musical la cual es confundible con una línea divisoria.

- Si una linea divisoria no es detectada correctamente el algoritmo no la tendría en cuenta y juntaría el compás con el siguiente.

En la figura 14 podemos observar el segundo problema, vemos que hay una nota blanca a la cual le falta el cuerpo, el resto del compás a partir de esa zona ha sido rellenado por el script para llenar la imagen, pero en realidad no existe. Un ejemplo de un compás bien recortado es la figura 15.

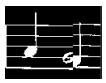

Fig. 14: Compás recortado erróneamente

Fig. 15: Compás bien recortado

Una posible solución es aplicar la transformada de Hough una vez se ha realizado morfología para quitar las líneas horizontales. De esta manera nos quedarían solo las verticales y sería mucho más sencillo recortar los compases.

5.2 Archivo MIDI

Antes de crear el archivo MIDI tenemos que leer el output del modelo y obtener los valores de los tokens. En el output tenemos para cada imagen que ha analizado el modelo sus correspondientes tokens. El archivo MIDI está formado por *tracks*, en la figura 15 podemos ver un fragmento del código, en concreto la parte donde se crea el archivo MIDI. Tomamos los valores del output y los añadimos a un *track* el cual añadimos al archivo MIDI. La figura 16 es el compás del cual estamos creando el MIDI. Con el siguiente enlace podemos escuchar como suena `https://bit.ly/3MBveJP`

```
vel = 0
pitch = 0
for key in notes:
    if str(key).startswith('delta'):
        delta_activate = True
        delta = int(''.join(map(str, notes[key])))
    if str(key).startswith('noteon'):
        vel = 90
        pitch = int(''.join(map(str, notes[key])))
        delta_activate = False
    if str(key).startswith('noteoff'):
        vel = 0
        pitch = int(''.join(map(str, notes[key])))
        delta_activate = False

    if not delta_activate:
        notesTrack.append(mido.Message('note_on', channel=0, note=pitch, velocity=vel, time=delta))
```

Fig. 16: Fragmento de código correspondiente a la creación del archivo .MIDI

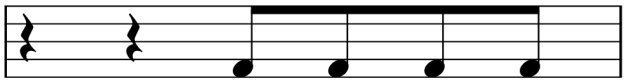

Fig. 17: Compás correspondiente al archivo .WAV

5.3 Entrenamiento con pentagramas

Se ha entrenado el modelo con el nuevo dataset de pentagramas, los resultados obtenidos han sido los siguientes:

TABLA 2: RESULTADOS CON EL NUEVO DATASET

Loss	42.52
SER	39.91%

Como podemos observar los resultados son peores cuando entrenamos el modelo con pentagramas en vez de compases. Mientras más largo es el pentagrama menos efectivo resulta el modelo, pues tiene que acordarse de mucha información. Las imágenes que generan mejores resultados son las de pentagramas que solo están formados por un compás, lo que se traduce directamente en una imagen de un compás como las que hay en el otro dataset. Al ser totalmente aleatoria la generación de los pentagramas, no es proporcional la cantidad de pentagramas que estén formados por más de un compás con los que solo tienen uno, pienso que es por esto que la función de pérdida y el SER dan unos valores muy altos.

5.4 Evaluación de los modelos

En este apartado vamos a evaluar los dos modelos, además de compararlos. El primer modelo ha entrenado con compa-

ses mientras que el segundo con pentagramas. Las métricas que vamos a usar son las siguientes:

- Symbol Error Rate (SER): proporción de elementos reconocidos erróneamente por el modelo.
- Función de pérdida: cálculo de la entropía cruzada.
- Precisión del tono: proporción de aciertos que el modelo ha obtenido en cuanto al tono de las notas.
- Precisión del tiempo: proporción de aciertos que el modelo ha tenido en cuanto a la duración de las notas.

La manera con la que medimos la precisión del tono y el tiempo es estricta. Miramos el porcentaje de aciertos que ha obtenido el modelo, si el modelo ha predicho más notas o menos, estas no son contadas.

Para este apartado, además de los resultados obtenidos con los datasets usados para entrenar, se han creado nuevos datasets con datos nunca vistos antes. Con estos vamos a poder comparar como se comportan los modelos con datos con diferente grado de dificultad a los que no están acostumbrados.

- Dataset 1: compases sencillos, formados por notas fáciles como negras, blancas, corcheas. Compases como los del dataset inicial.

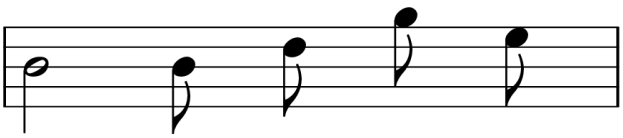

Fig. 18: Imagen del dataset 1

- Dataset 2: mezcla de compases y pentagramas sencillos.
- Dataset 3: compases con una dificultad elevada, hay tipos de notas no vistas antes por ninguno de los dos modelos. En la figura 17 podemos ver un ejemplo de esto. Vamos a entrenar más los dos modelos con este tipo de imagenes para ver como se comportan.

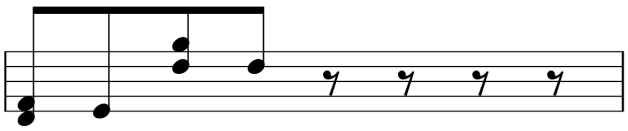

Fig. 19: Ejemplo de compás del dataset 3

Los resultados obtenidos de ambos modelos son los siguientes:

Observamos que obtiene mejores resultados en general el primer modelo. Clasificar compases suele resultar más sencillos que los pentagramas ya que estos tienen menos longitud y al modelo no le hace falta recordar tanta información. Podemos ver también que el segundo modelo, de las notas que ha podido identificar, tiene una precisión muy alta para el tiempo en comparación con el tono, esto puede

Métrica	Modelo 1	Modelo 2
Loss	0.36	42.52
SER (%)	0.3	39.91
Prec. tono	0.98	0.21
Prec. tiempo	0.99	0.86

TABLA 3: MÉTRICAS OBTENIDAS DE AMBOS MODELOS

ser porque son notas sencillas. No hay mucha variedad y los tiempos por el tipo de nota son fijos, es decir, una nota negra dura siempre lo mismo en cualquier pentagrama y posición. Pero el tono de la nota puede tomar valores muy distintos.

A continuación vamos a usar los datasets creados anteriormente para comparar a los dos modelos con diferentes datos. La figura 20 es una gráfica que los compara utilizando el Symbol Error Rate (SER).

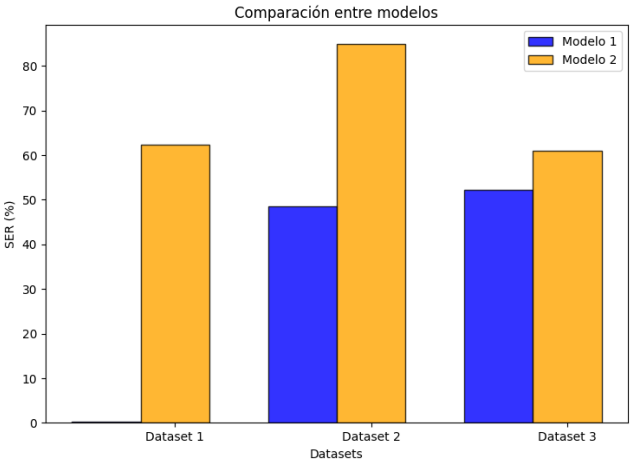

Fig. 20: Gráfica de los resultados obtenidos con los diferentes datasets

Para los distintos datasets podemos observar lo siguiente:

- Dataset 1: como era de esperar el primer modelo reconoce los elementos de los compases muy por encima de lo que puede el segundo modelo.
- Dataset 2: ambos modelos empiezan a reconocer erróneamente los elementos. En este dataset la mitad de los datos eran compáses y la otra mitad eran pentagramas, podemos observar que el modelo 1 ha aumentado el SER casi en un 50% y el modelo 2 casi un 30%.
- Dataset 3: este dataset ha hecho que el modelo 1 tenga el SER más alto con un 52% mientras que el modelo 2 ha podido bajar hasta un 61%.

Vemos pues, que ambos modelos trabajan mejor con imágenes de compases los cuales tienen la dificultad aumentada que con pentagramas enteros de varios compases. Esto se debe a que las imágenes de pentagramas pueden estar compuestas por más de 4 compases, haciéndolas muy extensas. Debido a esta longitud de la imagen, tras pasar por el pre procesado se redimensiona a un tamaño fijo de 256x800, haciendo que se pierdan muchos detalles de la imagen por lo que el modelo no es capaz de reconocer bien las figuras musicales.

Observemos cuales son los valores que han obtenido para la precisión del tono y del tiempo de las notas.

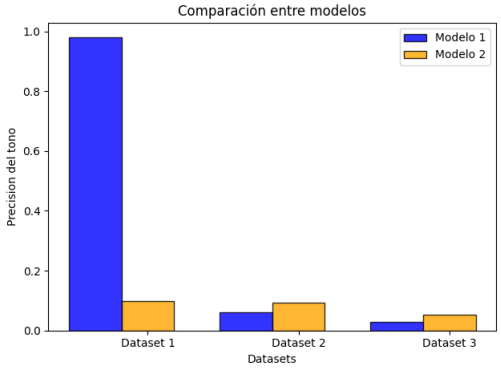

Fig. 21: Gráfica de los resultados obtenidos con los diferentes datasets para la precisión del tono

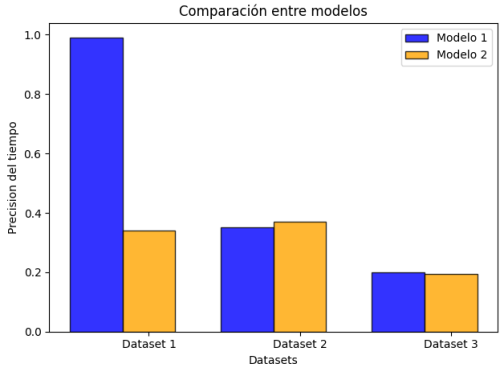

Fig. 22: Gráfica de los resultados obtenidos con los diferentes datasets para la precisión del tiempo

Podemos observar que el modelo da muchos mejores resultados cuando trabaja con partituras a nivel de compás. Entrenarlo con pentagramas empeora los resultados, pero podemos extraer de todo esto que encuentra más dificultades a la hora de clasificar correctamente el tono de las notas en lugar del tiempo.

6 Conclusión

A lo largo de este documento hemos podido observar el funcionamiento de todo el trabajo que se había lograda hasta la fecha. Se partía de una buena línea base, un algoritmo que genera partituras a nivel de compás con las cuales podemos generar la cantidad que queramos y crear nuestro propio dataset para entrenar el modelo, un dataset ya creado con aproximadamente 22 mil imágenes de compases y por último un modelo *sequence-to-sequence*.

El objetivo de este trabajo era entrenar el modelo con diferentes datos, mejorarlo, ver como se comportaba en diferentes situaciones,ver si nos podíamos ahorrarnos pasos del pre procesado y hacer que la salida generara el archivo MIDI correspondiente. Para lograr estos objetivos se ha cambiado el funcionamiento del algoritmo de partituras, haciendo que el modelo entrene con pentagramas enteros en vez de compases. La finalidad de este paso era ahorrarnos la parte del pre procesado de dividir los compases de los pentagramas ya que es un proceso complicado que puede fallar. Para el post procesado se ha creado un script que lee los output del modelo y crea el archivo MIDI correspondiente el cual podemos convertir en .WAV para escuchar como suena. Finalmente hemos visto una comparación de ambos modelos, el primero que entrenaba con compases y el segundo con pentagramas.

Viendo los resultados podemos concluir con que si bien entrenar el modelo con pentagramas nos podría ahorrar la parte del pre procesado, empeora muchos los resultados que se obtienen. Por último, si queremos que el modelo sea capaz de predecir correctamente el sonido de figuras más complejos habría que entrenarlo con estas también, ya que exponerlo a este tipo de imágenes sin haber entrenado con ellas antes hace que los resultados bajen.

Podemos concluir con que el modelo funciona mejor entrenando con partituras a nivel de compás, solo habría que trabajar sobre la parte de pre procesado y post procesado para hacer que funcionara con partituras musicales enteras.

Referències

[1] Calvo-Zaragoza, Jorge and Jr., Jan Hajič and Pacha, Alexander (2021, julio). Understanding Optical Music Recognition (N.o 4). Association for Computing Machinery (ACM). https://doi.org/10.1145/3397499

[2] Daniel Echevarría, ”Sistema de reconocimiento de partituras musicales y generación de archivos sonoros”. http://www.cvc.uab.es/people/afornes/students/TFG_2021_Daniel_Echevarria.pdf

[3] Jialuo Chen, https://bit.ly/3LzVTFJ

[4] Trello. (s. f.). Trello. Recuperado 2 de marzo de 2022, de https://trello.com/

[5] Hinchey, J. (2021, 28 diciembre). A review of optical music recognition software. Scoring Notes. https://bit.ly/3yRP4wp

[6] Shatri, E., & Fazekas, G. (2020). Optical Music Recognition: State of the Art and Major Challenges. arXiv. https://doi.org/10.48550/ARXIV.2006.07885

[7] A. Pacha, “Incremental supervised staff detection,” in 2nd International Workshop on Reading Music Systems, J. Calvo-Zaragoza and A. Pacha, Eds., Delft, The Netherlands, 2019, pp. 16–20.

[8] Rebelo, A., Capela, G., & Cardoso, J. S. (2009). Optical recognition of music symbols. En International Journal on Document Analysis and Recognition (IJDAR) (Vol. 13, Issue 1, pp. 19-31). Springer Science and Business Media LLC. https://doi.org/10.1007/s10032-009-0100-1

[9] van der Wel, E., & Ullrich, K. (2017). Optical Music Recognition with Convolutional Sequence-to-Sequence Models (Version 1). arXiv. https://doi.org/10.48550/ARXIV.1707.04877

[10] LilyPond (2.23.7). (2022). [Programa de edición de partituras]. http://lilypond.org/index.es.html

[11] Phi, M. (2020, 28 junio). Illustrated Guide to LSTM's and GRU's: A step by step explanation. Medium. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[12] CrossEntropyLoss — PyTorch 1.11.0 documentation. (s. f.). PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

[13] KLDivLoss — PyTorch 1.11.0 documentation. (s. f.). PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.KLDivLoss.html