Sistema de reconocimiento de partituras musicales y generación de archivos sonoros.

Daniel Echevarría Naharro

Resumen—El OMR o Reconocimineto Óptico de Música es una tecnología utilizada para el reconocimiento de partituras musicales a partir de imágenes para posteriormente procesarlas y crear un archivo de salida en formato de texto. Mi objetivo es utilizar un modelo ya utilizado en otros campos y adaptarlo para que la salida final sea un archivo de sonido reproducible sin pasos intermedios. Para ello, en este proyecto se utiliza un modelo Sequence to Sequence para generar a partir de una imagen de una partitura su correspondiente fichero de audio que posteriormente podrá ser tratado o editado. Los modelos Sequence to Sequence son un tipo de arquitectura de deep learning que han resultado dar muy buenos resultados en aplicaciones con reconocimiento de voz, traducción automática o descripción de videos entre muchos otros.

Palabras clave—seq2seq, deep learning, partitura musical

Abstract— The OMR or Optical Music Recognition is a technology used for the recognition of musical sheets from images to later process them and create an output file in text format. My objective is to use a model that has been already used in other fields and adapt it to make the output file into a reproducible sound archive with no intermediate steps. To achieve this, in this project it has been used a Sequence to Sequence model to generate, from a musical sheet image, a musical sheet audiofile which can be edited later. The Sequence to Sequence models are a type of deep learning architecture that give great results in applications such as voice recognition, automatic translation, or video description among others.

Index Terms—seq2seq, deep learning, music script

1 Introducción

El reconocimiento de partituras musicales a partir de imágenes es un campo muy activo y conitnuamente tratado. OMR o Reconocimiento Óptico de Música, es la tecnología utilizada para interpretar partituras manuscritas o impresas, procesarlas y crear un archivo de salida en formato editable o reproducible. El objetivo de estos proyectos usualmente se centra en facilitar la preservación de partituras clásicas, así como facilitar la edición de creaciones nunca editadas. En este caso, el objetivo varía. La intención del proyecto es crear una herramienta que facilite el aprendizaje músical. Dando, por ejemplo, a un estudiante la oportunidad de saber como suena la partitura que está viendo puede facilitar mucho su aprendizaje, sabiendo en todo momento si lo que está tocando corresponde con como realmente debería sonar.

En este proyecto el objetivo es, partiendo de partituras más básicas como las utilizadas en clases de primaria o escuelas de música (Fig. 1), centrar el esfuerzo en hacer que la salida de este proceso no sea únicamente el reconocimiento de la partitura, sino que se genere a partir de este reconocimiento un archivo reproducible de sonido que permita escuchar con exactitud esta misma partitura. De esta forma podríamos tener una idea de cómo debería de sonar esta sin necesidad de instrumentos ni de ayuda humana para tocarlos. Gracias a este procedimiento dotamos al usuario que quiera utilizar este algoritmo o cualquier otro proyecto derivado de este, de una facilidad a la hora de poder comprobar si la salida corresponde a la entrada con tan solo escuchar la pista de sonido. Este proceso diferencia a este proyecto de otros proyectos enfocados también en el OMR gracias a que se le da al usuario autonomía y se le evita tener que hacer procesos intermedios a la hora de convertir la salida a un formato entendible

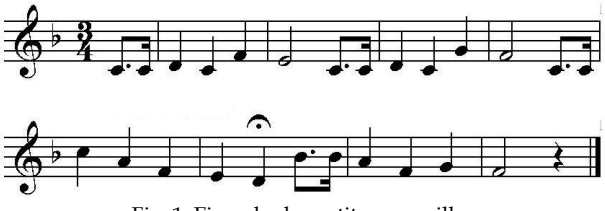


Fig. 1. Ejemplo de partitura sencilla.

- *E-mail de contacto: daniel.echevarria@e-campus.uab.cat*
- *Mención realizada: Enginyeria de Computació*
- *Trabajo tutorizado por: Alicia Fornes Bisquerra (Ciencias de la Computación)*
- *Curso 2020/21*

Para este objetivo, utilizaremos un tipo de arquitectura de deep learning llamada Sequence to Sequence (seq2seq). La cualidad principal de este tipo de modelos es que constan de dos módulos: un encoder y un decoder. Estos modelos son utilizados tanto en OMR como en OCR (Reconocimiento Óptico de Caracteres) ya que presentan una gran versatilidad dado que la entrada y la salida de estos modelos pueden ser diferentes tanto en tamaño como en naturaleza y logran dar buenos resultados en estos entornos. El encoder se encarga de comprender la entrada y el decoder de la salida. Esto puede resultar muy útil en distintos ámbitos entre los que podemos encontrar la traducción automática, el reconocimiento de voz, la descripción de imágenes y videos o un bot de chat entre otras muchas opciones. Por ejemplo, en el proceso de descripción de imágenes la entrada es una serie de pixeles y la salida un texto.

El trabajo que se ha realizado en este proyecto ha sido, en base a un algoritmo ya existente de reconocimiento de partituras manuscritas [1] cuyo objetivo y funcionalidad principal era a partir de la partirura impresa mostrar la salida en formato texto, realizar las modificaciones pertinentes para que, sin pasos intermedios de interpretación, la salida sea un fichero de audio.

El artículo está organizado de la siguiente forma:
Comenzaremos con una explicación de los objetivos planteados en este proyecto continuando con una explicación del estado del arte actual, tanto para los modelos de seq2seq como para algoritmos de generación de partituras, así como la metodología utilizada y la planificación establecida. En las siguientes secciones profundizaremos y expandiremos la información del funcionamiento de los algoritmos de generación y de deep learning mostrando el trabajo realizado. Para terminar, analizaremos los resultados obtenidos con su correspondiente conclusión general.

2 Objetivos

Tal como se ha explicado anteriormente, el objetivo principal de este trabajo consiste en lograr obtener un fichero de sonido a partir de la imagen de una partitura impresa. Para esto modificamos un algoritmo de reconocimiento de partituras para adaptarlo a nuestra finalidad y lograr que la salida sea conveniente con el trabajo que se necesitaba hacer. Con todo esto podemos definir los siguientes objetivos y subobjetivos:

1. Investigar y entender el funcionamiento de los algoritmos seq2seq, así como analizar otros ejemplos de seq2seq para comprender su funcionamiento.

2. Adecuar el algoritmo de generación de partituras para la creación de los distintos datasets y con el formato que deseamos para entrenar el modelo. Generar un conjunto de datos de prueba con los que trabajaremos y entrenaremos al modelo.

3. Modificar el algoritmo seq2seq inicial para adecuarlo a el reconocimiento de partituras impresas y la salida de audio reproducible.

4. Analizar e interpretar los resultados y realizar todas las pruebas necesarias.

3 Estado del arte

Con tal de analizar y comprender el panorama actual en la comunidad que trabaja en el desarrollo de OMR, se ha hecho un estudio sobre que alternativas tenemos en el mercado para cumplir el objetivo marcado en este proyecto.

Una de las aplicaciones más utilizadas y reconocidas en este entorno es Audiveris [2]. Esta herramienta de código abierto permite importar partituras escaneadas y exportarlas a un formato editable llamado MusicXML. Otra herramienta (está vez comercial) que se aproxima más a lo que se pretende desarrollar en este proyecto es SmartScore [3]. Esta aplicación permite reconocer las partituras, editarlas a tiempo real y reproducirlas. SmartScore también dispone de una aplicación Android y iOS que permite reproducir una partitura escaneada mediante el uso de la cámara o cargando una ya existente

Aunque el número de aplicaciones disponibles es bastante limitado (sobre todo de carácter gratuito), lo que si podemos encontrar es documentación dejada en forma de artículos o informes creados por otros investigadores. Esta documentación difícilmente incluye acceso al código desarrollado, pero resulta muy útil de por si ya que puede ayudar en la extracción de ideas para aplicar en este proyecto [4,5]. Entre estas herramientas de carácter científico o de investigación encontramos varias aplicaciones que utilizan modelos de Redes Neuronales Recurrentes (RNN) o Redes Neuronales Convolucionales (CNN) o incluso seq2seq como en nuestro caso, pero no hemos logrado encontrar ejemplos en los que la salida sea directamente el archivo de audio [6-12].

En cuanto a modelos de seq2seq con unas características similares a el objetivo de este artículo, podemos encontrar multitud de ejemplos a pesar de que la gran mayoría de ellos se centrar únicamente en el reconocimiento de la partitura en si y su transcripción a texto, dejando un poco de lado el enfoque que pretende este artículo de facilitar la reproducción de la partitura sin tener que realizar procesos intermedios.

4 Metodología

Para llevar a cabo este proyecto se ha aplicado una metodología ágil. Esta metodología proporciona un modelo iterativo e incremental susceptible a cambios y flexible para incorporar nuevas fases de ser necesario. Para lograr cada uno de los objetivos se realizarán reuniones semana-

les con la tutora para obtener un *feedback* sobre las tareas realizadas en la semana en curso y definir las próximas tareas a realizar. Esto permite tener un control sobre la evolución del proyecto y detectar los problemas que puedan surgir con antelación suficiente.

A partir de la metodología planteada, se ha realizado una planificación de las tareas a realizar a lo largo del trascurso del proyecto, que se pueden observar en el siguiente diagrama de Gantt (Figura 2).

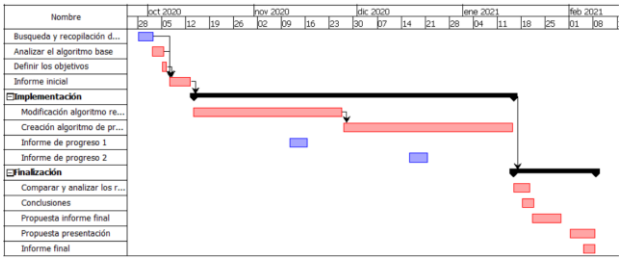


Fig. 2. Diagrama de Gantt.

El proyecto se ha dividido en distintas fases:

1. **Estudio del estado del arte**:

Esta es una de las fases más importantes del proyecto ya que se realizó una investigación para entender el contexto del problema y un análisis de las soluciones creadas por otros investigadores. También hacemos un estudio del algoritmo base para comprender el funcionamiento de este y ser capaz de realizar cambios en el para obtener el objetivo establecido. Una vez recopilada toda la información necesaria y habiendo analizado todo en profundidad, se valoró si los objetivos pensados inicialmente requerian ser revisados.

2. **Modificación/Implementación del algoritmo de generación de partituras**:

En esta etapa, tras estudiar toda la información recopilada en la fase anterior, se centró el esfuerzo en modificar el algoritmo para adecuar la generación de partituras para lograr generar las entradas y salidas deseadas.

3. **Modificación/Implementación del algoritmo seq2seq:**

Tras asegurar el formato de entrada y salida de el modelo, se procedió a la adaptación de este para que aceptara correctamente los ejemplos creados con el algoritmo de generación de partituras. Tras esto se trató de crear un archivo de sonido a partir de lo que el algoritmo ha conseguido reconocer de la imagen de la partitura impresa.

4. **Experimentación**.

Analizar los resultados obtenidos y tratar de mejorar la eficacia del algoritmo de ser necesario. Si los resultados no son favorables habría que replantear las dos fases anteriores.

5. **Documentación**.

Esto incluye los diferentes informes intermedios y la finalización del proyecto y redacción del artículo final, así como la creación de la presentación.

5 Algoritmo de generación de partituras

En esta sección el objetivo establecido fue generar una serie de conjuntos de imágenes y ficheros de sonido MIDI que funcionen como dataset con el que luego se procederá a entrenar el encoder del modelo seq2seq. La opción inicial era partir de un proyecto ya existente de reconocimiento y generación de partituras aleatorias, y adaptarlo para que además de la salida de un fichero de imagen también obtuviéramos el correspondiente fichero de sonido. Para esto, se pretendía modificar el funcionamiento del algoritmo de generación de partituras para que este además de la imagen generase también el correspondiente fichero MIDI. Con este fichero MIDI después se procedió a analizar posibles formas de transformarlo en información en formato escrito que pudiera ser utilizado en el entrenamiento del modelo. Finalmente, se concluyó que sería mucho más efectivo realizar un nuevo código (cogiendo varias ideas y conceptos del ya existente) para la generación de los primeros datasets sencillos, y una vez hechos, utilizar el algoritmo existente para crear ejemplos algo más complejos

Para la generación del fichero de imagen y el de sonido, se ha utilizado la librería Lilypond [8]. Entre las pocas alternativas que podemos encontrar hay herramientas online que directamente pueden generar según unos parámetros establecidos por el usuario, una partitura. El problema de estas herramientas está en que no tiene opción para generar un archivo de audio, y utilizarla para crear un dataset con múltiples elementos seria algo muy tedioso al tener que generar cada partitura una a una [9]. El código creado hace llamadas a esta librería con cadenas de caracteres aleatorias que simbolizan un conjunto de notas musicales, y esta devuelve una imagen con las notas representadas y un fichero MIDI de como sonarían. Los ficheros MIDI contienen información en formato hexadecimal. Entre esta información podemos encontramos la nota, el instrumento que la produce, en que secuencia, tempo, velocidad, tono e incluso información adicional como el nombre de la melodía o el autor [10]. Esta información se extraerá y nos permitirá entrenar el modelo para lograr que la salida tenga también este tipo de formato.

Además de esta cadena aleatoria, podemos manipular tanto manualmente como de forma aleatoria el tipo de notas que pueden aparecer según la duración (negra, blanca, semicorchea, corchea…) así como la posición de estas como hemos mencionado anteriormente. Mediante

una serie de parámetros realizamos la llamada a Lilypond que nos genera una imagen en formato PNG y un audio en formato MIDI (Fig. 3).

```
with open(filenameNotes1, 'w') as f1:
  f1.write(header)
  f1.write('#(set-global-staff-size 40)\n')
  f1.write("\\paper{\n")
  f1.write("top-margin = 0\n")
  f1.write("left-margin = 2\n")
  f1.write("right-margin = 2\n")
  f1.write("print-page-number = false\n")
  f1.write('#(set-paper-size "my size") oddFooterMarkup=##f\n')
  f1.write('#(define fonts\n')
  f1.write('(set-global-fonts\n')
  f1.write('#:factor (/ staff-height pt 22)\n')
  f1.write('))\n}\n')
  f1.write("\score {\n")
  f1.write("{\n")
  f1.write("\\hide Staff.Clef\n")
  f1.write("\\hide Staff.TimeSignature\n")
  f1.write("\\transpose c c' {\n")
  f1.write('\\override Staff.Clef.full-size-change = ##t\n')
  f1.write('\\accidentalStyle forget\n')
  f1.write('\\set Score.extraNatural = ##f\n')
  f1.write('\\set restNumberThreshold = #0\n')
  f1.write('\\autoBeamOff\n')
  f1.write('\\compressFullBarRests\n')
  f1.write('\\hide Score.BarNumber\n')
  f1.write('\\set restNumberThreshold = #0\n')
  f1.write('\\override MultiMeasureRest.expand-limit = #1\n')
  f1.write('\\override GraceSpacing.spacing-increment = #1.1\n')
  f1.write(hide_replaced_score)
  f1.write("} }\n")
  f1.write("\midi {}\n")
  f1.write("\layout {}\n")
  f1.write("}\n")

call(['lilypond', '--output=midi', '-ddelete-intermediate-files',
      '--loglevel=WARN', '-fpng', '-dresolution=200', filenameNotes1])
```

Fig. 3. Extracción del código de generación de partituras aleatorias. Sección en la que se crea el archivo utilizado en la librería Lilypond para la creación de la salida en formato imagen y MIDI de la partitura seleccionada en el string *hide_replaced_score*.

En términos más específicos, el código comienza generando una cadena de texto concatenando el tipo de nota, la octava y su posición en el pentagrama de forma aleatoria según las opciones definidas por el usuario (Fig. 4). Estos datos se guardan en un fichero junto a una serie de parámetros adicionales que permiten establecer el formato que tendrán las salidas tras la llamada a Lilypond o directrices sobre el estilo de la imagen generada (tamaño, margen, esconder la clave y el compás…).

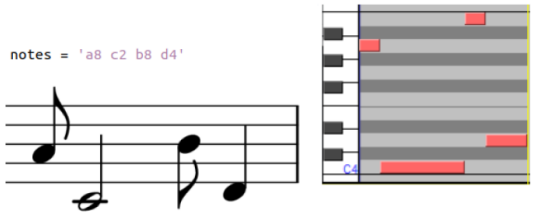


Fig. 4. Ejemplo de segmento de partitura con combinación de notas en cadena de texto junto a su respectiva interpretación impresa y su representación en formato MIDI (interpretado en Audacity).

6 CREACIÓN DEL DATASET

En esta sección que correspondería a la segunda etapa del proyecto, el objetivo establecido fue comenzar el entrenamiento del modelo mediante una estructura seq2seq. Inicialmente el objetivo era únicamente realizar las pruebas con el modelo, generando los archivos de sonido como salida, pero esto ha evolucionado a una nueva idea dado que podría resultar interesante, entrenar dos modelos, uno de ellos con su salida en formato de texto y otro con la salida en audio, ambos con el mismo dataset de imágenes de entrada. De esta forma ganaríamos la capacidad de poder realizar comparaciones con ambos resultados viendo la efectividad y diferencia de cada uno de estos modelos o si por lo contrario la diferencia en los resultados es mínima. Con este plan en mente, se procedió a intentar adaptar el código existente a nuestra necesidad.

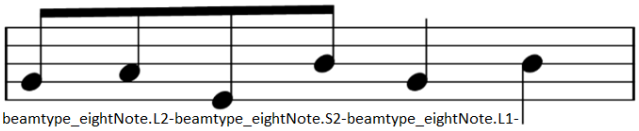
Para este trabajo, se generó un dataset con más de 20.000 muestras aleatorias con su correspondiente salida tanto a formato de texto como a formato MIDI. A partir de este dataset se comenzará a entrenar el modelo con el algoritmo de generación de partituras aleatorias comentado anteriormente. El dataset se dividió en tres subdatasets, uno para el entrenamiento, otro para la validación y uno último para el testeo. Las imágenes de este dataset constan de imágenes con fragmentos de partituras, entre las que solo pueden hallarse redondas, blancas, negras, corcheas, semicorcheas y silencios. En el intento de implementación de los cambios necesarios para hacer funcionar el código con el dataset y el objetivo propio, surgieron multitud de dificultades. La complejidad del algoritmo es alta, y llegar a entender cómo funciona todo teniendo un proyecto tan extenso puede ser una tarea compleja.

Tras multitud de intentos se logró hacer funcionar con un dataset de ejemplo proporcionado por el creador del propio código [11] y, tras esto, se comenzaron a implementar las adaptaciones necesarias para nuestro propósito. A partir de estas modificaciones, se realizaron pruebas modificando las entradas y salidas del algoritmo a nuestras necesidades para ver si este funcionaba sin dificultades y los resultados eran los esperados para un dataset de este tamaño reducido.

Como hemos mencionado anteriormente, para realizar las diferentes pruebas con el algoritmo seq2seq, entrenamos dos modelos distintos con el mismo conjunto de entrada, pero un conjunto de salida distinto, uno de ellos en formato de texto y el otro en formato reproducible.

- Dataset A - Salida en formato de texto:

Las salidas de este dataset se componen por una serie de labels que clasifican lo que contiene la imagen de la partitura siguiendo un determinado orden de lectura (de izquierda a derecha) como se muestra en la Figura 5. Estos labels indican tanto el tipo de nota (corchea, semicorchea, negra, blanca…) como la posición en la que se encuentra cada nota.

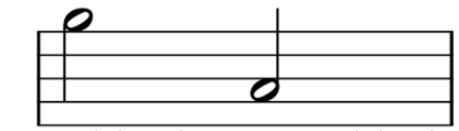


beamtype_eightNote.L2-beamtype_eightNote.S2-beamtype_eightNote.L1-
beamtype_eightNote.L3-notype_quarterNote.L2-notype_quarterNote.L3

Fig. 5. Ejemplo de partitura con su correspondiente salida en formato de texto.

▪ Dataset B - Salida en formato de audio:

En este dataset encontramos de la misma forma que en el anterior un fichero de texto que contiene distintos datos extraídos de los correspondientes ficheros MIDI. Estos han sido previamente generados juntamente a la creación de partituras aleatorias y que serán posteriormente convertidos a archivos de audio reproducibles. Para la creación de estos ficheros MIDI requerimos de tres parámetros que podemos apreciar en la Figura 6: nota, velocidad y tiempo. Tanto para obtener estos datos a partir de un MIDI ya existente como para generar un nuevo fichero, utilizaremos Mido [12]. Esta librería está diseñada para facilitar el trabajo y edición de ficheros MIDI. De esta forma podemos extraer de forma fácilmente leíble la información que contienen estos ficheros y crear los nuestros propios utilizando los parámetros explicados anteriormente.

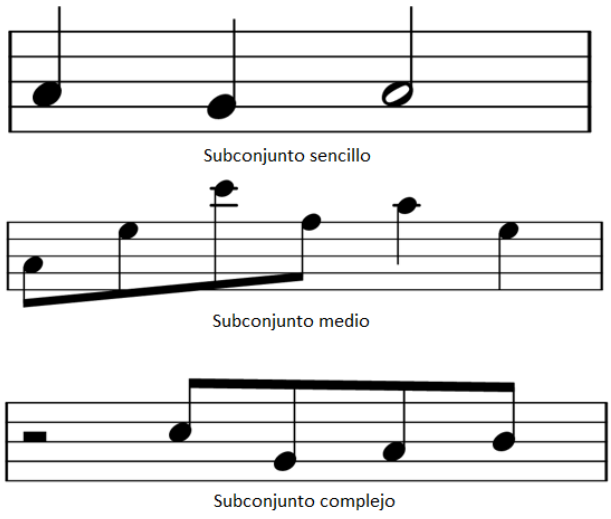


note=79.velocity=90.time=1536-note=79.velocity=0.time=768-
note=69.velocity=0.time=90-note=69.velocity=0.time=768

Fig. 6. Ejemplo de partitura con su correspondiente salida en formato de texto.

Mido utiliza una estructura con formato de mensajes para informar los distintos campos necesarios a la hora de crear un MIDI. Gracias a estos mensajes podemos no solo customizar las notas y a que velocidad o tiempo se van a reproducir, si no que también podemos informar de en que clave se deben reproducir, el tempo o incluso el nombre del audio y el autor o comentarios varios.

Para realizar el análisis del comportamiento del modelo seq2seq en diferentes situaciones, utilizaremos tres subconjuntos de datos como datasets, aumentando la complejidad de estos en comparación al dataset anterior. De esta forma, el primer dataset únicamente contendrá notas muy sencillas, entre las que podemos encontrar negras, blancas y redondas. Para el segundo dataset, añadiremos a las partituras más variedad de notas encontrando además de las mencionadas anteriormente corcheas y semicorcheas. Para terminar, el tercer y más complejo dataset además de todo esto mencionado, incluirá silencios de negra, blanca, corchea y semicorchea (Fig. 7).



Subconjunto sencillo

Subconjunto medio

Subconjunto complejo

Fig. 5. Ejemplo de los diferentes subconjuntos.

7 ALGORITMO SEQ2SEQ

Usualmente los algoritmos de seq2seq funcionan de una forma muy similar, aunque puede haber variaciones. Este algoritmo se podría dividir en la fase de entrenamiento y en la fase de inferencia.

▪ Fase de entrenamiento:

En esta fase el proceso consiste en entrenar el encoder y el decoder (Fig. 8). El encoder utilizando una serie de unidades o nodos recurrentes y dada una entrada, extrae puntos claves o características de esta para luego propagarla a la siguiente unidad. El decoder funciona de forma similar al encoder. Este es entrenado con una serie de muestra e intenta predecir la salida de cada nodo según la salida del anterior. De esta forma el encoder se encarga de comprender la forma y naturaleza de la entrada y el decoder de la salida.

▪ Fase de inferencia:

En esta fase se realizan pruebas con el modelo entrenado, utilizando nuevas entradas para ver cómo se comporta. Para ello, se le pasa al encoder las nuevas entradas y se inicializan los valores internos del decoder. Con cada iteración, el decoder calcula las probabilidades de las posibles siguientes palabras y escoge como palabra la qual tenga una mayor probabilidad. A continuación, se le pasa al decoder la palabra prevista en la anterior iteración, se actualizan los valores internos y se repite el proceso. Esto se realiza hasta llegar al final de la secuencia.

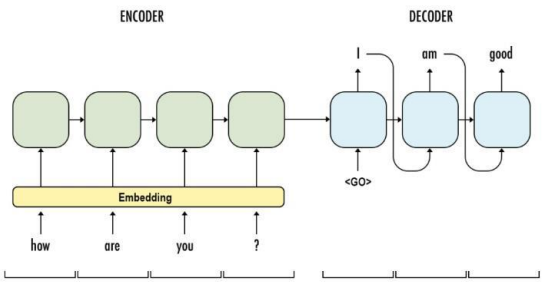


Fig. 8. Estructura de modelo seq2seq para la traducción automática.

Aplicando este funcionamiento a nuestros datasets, logramos entrenar al encoder con una serie de partituras impresas y una serie de salidas en forma de texto o audio que representan a la imagen de entrada para posteriormente, validar si después del entrenamiento el modelo es capaz de clasificar correctamente las nuevas muestras introducidas.

Además de todo esto, el objetivo principal de este proyecto era lograr que la salida del modelo seq2seq fuera un archivo de audio reproducible. Para ello, comenzamos recomponiendo a partir del fichero de predicciones los diferentes archivos MIDI utilizando Mido. Después de generar este archivo, utilizando la librería midi2audio [13] convertimos el MIDI recién creado a un archivo WAV que podemos reproducir con total facilidad (Fig. 9).

```
for aux in newlines:
    mid = MidiFile()
    track = MidiTrack()
    mid.tracks.append(track)
    track.append(MetaMessage('set_tempo', tempo=1000000))
    aux = aux.split("~")
    for word in aux:
        auxword = word.split(".")
        for letter in auxword:
            if count == 0:
                vnote = (letter[letter.rfind("="):]).replace("=","")
            if count == 1:
                vvelocity = (letter[letter.rfind("="):]).replace("=","")
            if count == 2:
                vtime = (letter[letter.rfind("="):]).replace("=","")
            count = count + 1
        count = 0
        track.append(Message('note_on', note=int(vnote),
                             velocity=int(vvelocity), time=int(vtime)))
    mid.save('midi/'+str(filename[auxcount])+'_syn_beethoven.mid')
    fs = FluidSynth()
    fs.midi_to_audio('midi/'+str(filename[auxcount])+'_syn_beethoven.mid',
                     'wav/'+str(filename[auxcount])+'_syn_beethoven.wav')
```

Fig. 9. Código encargado de la creación del fichero de audio.

7 Resultados

En esta sección se realiza un estudio de los resultados extraidos del algoritmo a partir de diferentes métricas. También realizaremos una comparación entre los resultados dados por cada uno de los modelos entrenados con diferentes datasets de forma que podamos analitzar si ademas de la comodidad de obtener directamente un archivo reproducible, este genera resultados suficientemente buenos y confiables.

Para comenzar el análisis de diferentes situaciones, utilizaremos los tres subconjuntos de datos mencionados anteriormente. Para evaluar los distintos resultados nos fijaremos en el SER (Symbol Error Rate) que simboliza la cantidad de elementos que no han sido reconocidos correctamente por el modelo. Esto se puede calcular mediante la siguiente formula:

$$SER = \frac{Inserciones + Borrados + Substituciones}{Número\ de\ elementos}$$

Con esto mencionado, los resultados son los siguientes mostrados en la Tabla 1.

	Dataset A	Dataset B	Dataset C
Modelo 1 (Salida Texto)	16,12%	21,82%	57,56%
Modelo 2 (Salida Audio)	11,02%	18,81%	52,55%

Tabla 1. Resultados de los diferentes datasets en ambos modelos.

Como podemos apreciar, subiendo la complejidad del algoritmo encontramos que el SER aumenta, indicando que la precisión del modelo disminuye a su mismo tiempo (Fig. 10).

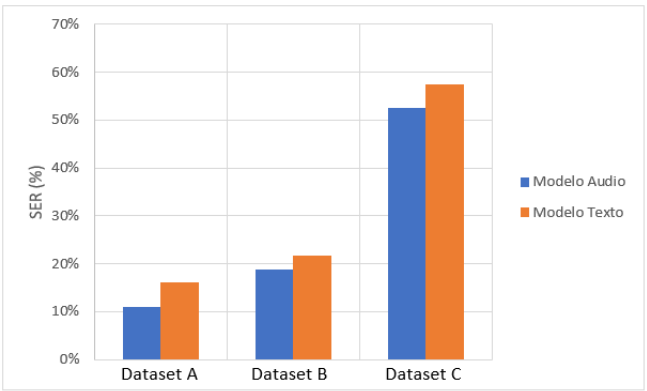


Fig. 10. Ilustración gráfica de los datos obtenidos en la Tabla 1.

A continuación, analizaremos dataset a dataset cuales son los puntos que influyen en este rendimiento y posibles soluciones que podrían aplicarse para mejorar los resultados.

- Dataset A:

Como hemos mencionado anteriormente, este dataset cuenta de partituras lo más sencillas posibles. Encontramos entre las posibles notas únicamente redondas, negras y blancas. De esta forma, el resultado que obtenemos es bastante bueno a pesar de que hay cierto margen de error, ya que el número de partituras tan sencillas que hemos utilizado para entrenar el modelo es muy bajo, siendo la gran mayoria de una complejidad algo superior. Esto hace que tenga peor rendimiento de lo que debería, pero seria fácilmente solucionable entrenando el algoritmo con un subconjunto moderado de partituras de dicha simpli-

cidad. Como podemos ver (y además se repetirá en los siguientes datasets) los resultados del modelo de salida en formato texto tiene un rendimiento ligeramente inferior al modelo de salida de audio. Después de analizar cada dataset por separado se analizará esta situación con más detalle.

▪ Dataset B:

En este caso, el dataset contiene partituras que podríamos considerar más completas. Entre las opciones de notas encontramos redondas, blancas, negras, corcheas y semicorcheas. La precisión del modelo disminuye un poco, aunque no de manera drástica. Como en el caso del dataset anterior, encontramos que el algoritmo resulta más efectivo con el dataset de salida en formato de audio. En este caso, una forma de mejorar los resultados y hacer el modelo aún más efectivo seria simplemente continuar entrenando el modelo con un conjunto mayor de partituras a lo utilizado actualmente. A pesar de todo esto, el resultado en este caso es más que favorable.

▪ Dataset C:

Para finalizar este análisis de los datasets, encontramos el dataset C. Este conjunto de partituras incluye además de las notas mencionadas anteriormente silencios de negras, blancas, redondas, corcheas y semicorcheas. Como apreciamos en los resultados, en este caso el SER aumenta bastante en comparación al resto de datasets, haciendo que obtengamos una precisión de poco más de un 45% de media entre ambos modelos. Tras analizar posibles causas de esto, se observó que el dataset que se utilizó para entrenar el modelo no contenía suficientes ejemplos de silencios como para poder obtener un resultado óptimo dentro de un conjunto de entrenamiento tan grande. Como en los casos anteriores, sería fácilmente solucionable disponiendo de más tiempo y recursos para poder efectuar el entreno del módelo con un conjunto mayor, que incluya más casos que ayuden a mejorar la precisión de este tipo de partituras.

Para terminar, compararemos los resultados obtenidos en los diferentes modelos, tratando de analizar la causa de que el modelo con salida en formato de texto tenga peor rendimiento que el de salida de audio. Tras diversas pruebas y análisis llegamos a la conclusión de que la razón más probable de esto estaba en las posibles clasificaciones de las notas. En el modelo de salida de texto, aunque el formato fue simplificado, encontramos que el formato de las labels es bastante complejo como podemos apreciar en la Fig. 10.

```
'notype_eighthNote.S4', 'beamtype_eighthNote.S4',
'notype_halfNote.S4', 'beamtype_halfNote.S4',
'notype_quarterNote.S4', 'beamtype_quarterNote.S4',
'notype_wholeNote.S4', 'beamtype_wholeNote.S4',
```

Fig. 10. Ejemplo de labels del modelo con salida de texto.

Por otro lado, en el caso del modelo de salida de audio, encontramos que, aunque las opciones de labels son mucho mayores, todo oscila entorno a tres variables diferentes: *note*, *velocity* y *time* como hemos mencionado anteriormente en el articulo. Cada una de estas variables puede tener un distinto número de valores; por ejemplo, el valor de la nota puede ser un *integer* entre el 0 y el 127 pero tanto la velocidad como el tiempo tienen unas opciones más reducidas (Fig. 11). Tras analizar esto, mi conclusión es que el modelo tiene más facilidad a la hora de clasificar un elemento si los labels de comparación son parámetros establecidos entre un cierto rango de números en lugar de una serie de caracteres de texto que son más variables y es más complejo que se repitan en una misma secuencia a lo largo de un dataset, haciendo que haya un menor número de posibles clasificaciones.

```
'note=14.velocity=0.time=0', 'note=14.velocity=0.time=192', 'note=14.velocity=0.time=384',
'note=14.velocity=0.time=768', 'note=14.velocity=0.time=1536', 'note=14.velocity=90.time=0',
'note=14.velocity=90.time=192', 'note=14.velocity=90.time=384', 'note=14.velocity=90.time=768',
'note=14.velocity=90.time=1536', 'note=15.velocity=0.time=0', 'note=15.velocity=0.time=192',
'note=15.velocity=0.time=384', 'note=15.velocity=0.time=768', 'note=15.velocity=0.time=1536',
'note=15.velocity=90.time=0', 'note=15.velocity=90.time=192', 'note=15.velocity=90.time=384',
'note=15.velocity=90.time=768', 'note=15.velocity=90.time=1536'
```

Fig. 11. Ejemplo de labels del modelo con salida de audio.

A modo de resumen, los resultados más beneficiosos de este proyecto han resultado ser en el modelo con salida en formato audio, coincidiendo con el objetivo del proyecto en si. Ambos modelos coinciden con bastante exactitud en las entradas que se predicen incorrectamente. Estas partituras suelen ser con una serie de notas consecutivas en posiciones muy similares y del mismo tipo, lo que hace que falle en contadas ocasiones al intentar predecirlas (Fig. 12).

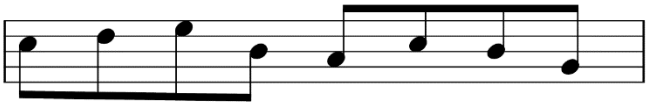


Fig. 12. Ejemplo de partitura predecida incorrectamente.

8 Conclusión

En este trabajo el objetivo era lograr procesar una imagen de una partiturara para obtener un fichero de audio de como sonaría, para ello hemos llevado a cabo varios estudios que después hemos combinado entre si para obtener una salida concreta. Gracias a los conocimientos ya adquiridos sobre el entreno y funcionamiento de modelos de *deep learning* y el tratamiento y reconocimiento de elementos en imágenes, añadiendo lo aprendido que se ha logrado adquirir gracias al estudio de diferentes algoritmos y artículos, hemos logrado extraer resultados y lograr el objetivo satisfactoriamente. Como hemos analizado en la sección de resultados, aunque el resultado final está lejos de ser óptimo resulta muy prometedor y anima a seguir trabajando en esto.

En cuanto a el modelo seq2seq, esta parte ha sido la más sencilla de implementar, aunque la más difícil de comprender. Las pinceladas de conocimiento de modelos de *deep learning* estaban lejos de ser suficientes para el entendimiento de lo que se pretendía hacer, por lo que se ha requerido un gran esfuerzo y recaudación de información para comprender todo perfectamente. Inicialmente el objetivo era generar directamente un archivo de audio en lugar de un MIDI, pero posteriormente se concluyó que podría ser interesante hacer como paso intermedio esta generación a MIDI. Los archivos MIDI son mucho más sencillos de tratar y permitirán facilitar el entrenamiento del modelo seq2seq. Durante el análisis de este código y la búsqueda de información del funcionamiento de los ficheros MIDI, surge una nueva opción: una serie de datasets ya creados que relacionan partitura con archivo de sonido que podrían acortar el tiempo de generación de partituras al hacer un dataset conjunto [14]. Tras leer detenidamente el articulo y observar con detención los diferentes datasets, llegamos a la conclusión de que hay que descartar esta opción debido a que tanto las imágenes como los archivos de sonido describen una partitura completa y generalmente compleja que no ayudarían en el entrenamiento inicial de nuestro modelo seq2seq.

Por otro lado, hubiera sido interesante disponer de mayor tiempo para poder duplicar el esfuerzo y buscar opciones de expansión de estos algoritmos para poder utilizar a tiempo real, como podría ser una pequeña pagina web o aplicación Android o iOS que con el modelo ya entrenado y al indicarle una imagen de entrada, te devolviera su correspondiente fichero reproducible de salida. Además, hubiera sido también interesante poder entrenar el modelo para agregar partituras mucho más complejas que pudieran contener acordes, y ver los diferentes resultados dados con esto.

Todo esto queda como una opción a realizar en un futuro ya que considero que es un tema muy interesante del que podrían resultar proyectos muy fructíferos.

AGRADECIMIENTOS

Para resumir, decir que ha sido un placer poder trabajar en este proyecto y por ello lo primero es dar gracias a mi tutora, Alicia Fornes. Siempre he querido ampliar más mis pocos conocimientos musicales, y esta ha sido una muy buena oportunidad para ello. De la misma manera, me he visto en una situación en la que también he aprendido muchísimo sobre el funcionamiento de los modelos de *deep learning* que considero no hubiera podido aprender de otra forma mejor. Por último, gracias a mi pareja por el apoyo en todos los momentos de estrés y dificultades durante el trabajo (que no han sido pocos), sin ti no hubiera encontrado la motivación.

BIBLIOGRAFIA

[1] Arnau Baró. "Handwritten Historical Music Recognition by Sequence-to-Sequence with Attention Mechanism". http://www.cvc.uab.es/people/abaro/papers/2020_ICFHR_ABaro.pdf
[2] Audiveris. https://audiveris.github.io/audiveris/
[3] SmartScore. https://www.musitek.com/
[4] Julia Adamska, Mateusz Piecuch, Mateusz Podgórski, Piotr Walkiewicz, Ewa Lukasik. Mobile System for Optical Music Recognition and Music Sound Generation. 14th Computer Information Systems and Industrial Management (CISIM), pp. 571 - 582, 2015. https://hal.inria.fr/hal-01444498/
[5] Eelco van der Wel, Karen Ullrich "Optical Music Recognition with Convolutional Sequence-To-Sequence Models". https://arxiv.org/pdf/1707.04877.pdf
[6] Zhiqing Huang, Xiang Ji and Yifan Guo "State-of-the-Art Model for Music Object Recognition with Deep Learning", 2019. https://www.mdpi.com/2076-3417/9/13/2645
[7] Liang Chen, Rong Jin, Simo Zhang, Stefan Lee, Zhenhua Chen, David Crandall. "A hybrid HMM-RNN model for optical music recognition", Extended abstracts for the Late-Breaking Demo Session of the 17th International Society for Music Information Retrieval Conference, 2016. https://wp.nyu.edu/ismir2016/wp-content/uploads/sites/2294/2016/08/chen-hybrid.pdf
[8] A.Baró, P.Riba, J.Calvo-Zaragoza, A.Fornés. Optical Music Recognition by Long Short-Term Memory Networks. Graphics Recognition, Current Trends and Evolutions, Lecture Notes in Computer Science, vol.11009, pp. 81 - 95, A.Fornés, B.Lamiroy (Eds), Editorial: Springer, Cham, ISBN 978-3-030-02283-9, 2018.
[9] J Calvo-Zaragoza, JJ Valero-Mas, A Pertusa. END-TO-END OPTICAL MUSIC RECOGNITION USING NEURAL NETWORKS. International Society for Music Information Retrieval Conference, 2017 https://simssa.ca/assets/files/jorge-ismir2017.pdf
[10] A Pacha, J Hajič, J Calvo-Zaragoza. A baseline for general music object detection with deep learning. Applied Sciences, 2018
[11] A Baró, P Riba, J Calvo-Zaragoza, A Fornés. From optical music recognition to handwritten music recognition: A baseline. Pattern Recognition Letters, 2019
[12] L. Tuggener, I. Elezi, J. Schmidhuber, and T. Stadelmann, "Deep watershed detector for music object recognition," in ISMIR, 2018, pp. 271 – 278
[13] Lilypond. http://lilypond.org/index.es.html
[14] Sheet Music Generator http://www.randomsheetmusic.com/
[15] Standard MIDI-File Format Spec. http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html
[16] Arnau Baró. "Old Music Scores by Pau Llinàs". http://www.cvc.uab.es/people/abaro/datasets.html
[17] Mido - MIDI Objects for Python. https://mido.readthedocs.io/en/latest/index.html#mido-midi-objects-for-python
[18] midi2audio. https://pypi.org/project/midi2audio/
[19] Frank Zalkow, Stefan Balke, Vlora Arifi-Müller, Meinard Müller "MTD: A Multimodal Dataset of Musical Themes for MIR Research", 2020 https://transactions.ismir.net/articles/10.5334/tismir.68/