UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Digitalización de placas astronómicas antiguas

AUTORES: Ponte Ahón Santiago Andrés

DIRECTOR/A: Dr. Ronchetti Franco

CODIRECTOR/A: Dr. Quiroga Facundo

ASESOR/A PROFESIONAL:

CARRERA: Licenciatura en Sistemas

Resumen

La FCAG (Facultad de Ciencias Astronómicas y Geofísicas de la Universidad Nacional de La Plata) posee una vasta colección de placas astronómicas antiguas las cuales contienen una enorme cantidad de registros astronómicos únicos y de alto valor histórico. Estas están en un formato de vidrio inadecuado para su uso con tecnologías modernas, por lo que se busca digitalizarlas según los estándares actuales. En este trabajo se desarrolló un software que ayuda con el proceso de digitalización, reduciendo en gran medida los tiempos necesarios para digitalizar cada placa. La aplicación desarrollada demostró ser una opción eficaz para agilizar el proceso de digitalización e incorpora de forma exitosa los sistemas previamente mencionados.

Palabras Clave

Tesina de grado, placas espectrográficas, aprendizaje automático, detección de objetos, obtención de metadatos, digitalización de material físico.

Trabajos Realizados

Inicialmente, se realizó un estudio detallado del problema a resolver, se identificaron las partes esenciales del proceso de digitalización de placas espectrográficas y se definió como la aplicación a desarrollar podría ayudar con la realización de cada una. Luego, se diseñó la interfaz y se procedió con la construcción del software, se definió un Backend para resolver las tareas de procesamiento complejas y un Frontend centrado en tratar con la interacción del usuario. Entre otras funciones, el software asiste al usuario en el recorte e identificación de espectros, para ello utiliza un modelo de detección de objetos YOLO que fue entrenado con un conjunto de datos recopilado y etiquetado para la ocasión. Por último, se testeo el funcionamiento del software con varios astrónomos profesionales, para detectar bugs y mejorar sus características.

Conclusiones

Se pudo desarrollar el sistema propuesto y en las pruebas que se hicieron con astrónomos profesionales el software demostró su capacidad de reducir los tiempos de digitalización promedio, logrando digitalizar un escaneo cada aproximadamente 15 minutos, respecto a la hora que esta tarea suele tardar. Esta reducción mejora significativamente los tiempos estimados para la digitalización de grandes colecciones de placas espectrográficas, tales como la que posee la Facultad de Ciencias Astronómicas y Geofísica de La Plata, con una persona digitalizando a jornada completa se estima que podría tardar 8 años en digitalizar toda la colección, pero haciendo uso del software esta cifra se podría reducir a 2.

Trabajos Futuros

Se identificaron como trabajos futuros: continuar optimizando la eficiencia de carga de información en pos de reducir aún más los tiempos de digitalización, automatizar las tareas posteriores a la digitalización de las placas espectrográficas ya sea realizar el análisis de las lámparas de comparación o el análisis de los espectros, actualizar el software para que sea capaz de atender múltiples usuarios de forma segura, y la posibilidad de incorporar al sistema el uso de un DBMS en caso de que en un futuro se quiera resolver tareas de indexación y búsqueda .

Fecha de la presentación: Marzo de 2023

Universidad Nacional de La Plata

Facultad de Informática

Digitalización de placas astronómicas antiguas

Tesis de grado

Director: Dr. Ronchetti Franco
Codirector: Dr. Quiroga Facundo
Alumno: Ponte Ahón Santiago Andrés

Marzo 2023

Agradecimientos

Principalmente, a mi familia por su apoyo incondicional en lo que sea que haga.

A la gente del LIDI por ayudarme y animarme durante todo el proceso de escritura de tesina.

A mis directores Franco Ronchetti y Facundo Quiroga cuyas pacientes y carismáticas correcciones hicieron posible este trabajo.

A mis contactos del proyecto ReTrOH Roberto Gamen y Yael Aidelman por resolver todas mis dudas del ámbito astronómico.

A Joaquín Miranda por trabajar conmigo durante mis primeros días en el laboratorio.

A mis amigos de la Facultad de Informática por acompañarme durante todo este tiempo.

Índice general

Capítulo 1

Introducción

1.1. Resumen

La FCAG (Facultad de Ciencias Astronómicas y Geofísicas de la Universidad Nacional de La Plata) posee una vasta colección de placas astronómicas antiguas las cuales contienen una enorme cantidad de registros astronómicos únicos y de alto valor histórico. Estas están en un formato de vidrio inadecuado para su uso con tecnologías modernas, por lo que se busca digitalizarlas según los estándares actuales. En este trabajo se desarrolló un software que ayuda con el proceso de digitalización, reduciendo en gran medida los tiempos necesarios para digitalizar cada placa. La totalidad del desarrollo consistió en 4 partes:

1. Generación de un modelo que pueda recortar los espectros individuales de los escaneos de cada placa haciendo uso de tecnologías de Aprendizaje Automático.

2. Solucionar la interacción con bases de datos astronómicas, para que faciliten los metadatos con los que configurar los archivos FITS resultantes.

3. Desarrollo de un backend que incorpore estos dos sistemas y los mantenga disponibles para su utilización por terceros en forma de API REST.

4. Desarrollo de un Frontend que en base a la interacción con los sistemas antes descriptos permita a un usuario subir escaneos de placas astronómicas, recortar los espectros que contiene, que asista al usuario en el proceso de digitalización con el fin de cargar y validar los datos de cada placa en el menor tiempo posible.

La aplicación desarrollada demostró ser una opción eficaz para agilizar el proceso de digitalización e incorpora de forma exitosa los sistemas previamente mencionados. En la figura 1.1 se puede ver un diagrama explicativo de su funcionamiento.

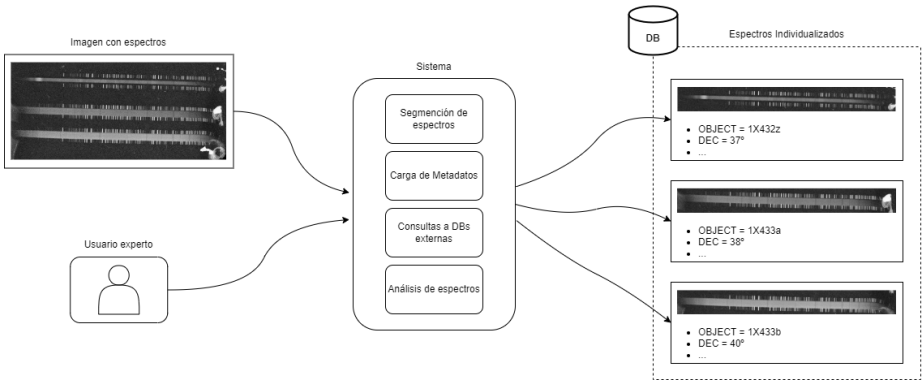

Figura 1.1: Diagrama del funcionamiento del software generado

1.2. Motivación

La FCAG posee una importante colección de placas espectrográficas, las cuales guardan una enorme cantidad de información única y de alto valor histórico que mediante su análisis puede posibilitar nuevos descubrimientos. Históricamente, el procesamiento de este tipo de placas se realiza mediante métodos puramente manuales, que ofrecen excelentes resultados pero requieren dedicar una considerable cantidad de tiempo, que se estima en una hora por placa. Dada la colección de miles que se posee, una persona trabajando a diario 8 horas al día, terminaría de procesar la colección en aproximadamente 8 años. Afortunadamente, la mayoría de las placas todavía están en un estado de conservación aceptable, pero su calidad no hará mas que decrementar con el paso del tiempo por lo que es prioritario hallar un método que asegure la conservación de su información lo antes posible.

La captura de información astronómica en placas espectrográficas fue un método popular en su tiempo, por lo que muchas organizaciones se encuentran en situaciones similares, disponiendo de colecciones masivas de placas astronómicas mas allá de lo que son capaces de procesar. Esta situación no sería tan problemática si las placas simplemente se pudiesen mantener almacenadas de forma indefinida. No obstante como se mencionó antes, las placas son vulnerables al paso del tiempo, por lo que mientras más tiempo se las mantienen almacenadas, más información se termina perdiendo. Considerando esta situación la UAI (Unión Astronómica Internacional) emitió una resolución en el año 2000 donde solicita la conservación de este tipo de placas para que puedan ser aprovechadas por futuras generaciones de astrónomos [UAI, 2000].

En Argentina ya se han establecido iniciativas que procuran la conservación y procesamiento de estas placas, como el proyecto ReTrhOH [ReTrhOH, 2019] que tiene como objetivo recuperar y poner en valor el trabajo observacional, en el que estas placas están incluidas. También hay trabajos que ayudan en este propósito como la tesis de Natalia Soledad Meilán, en la cual se digitalizan

y analizan algunos elementos de la colección antes mencionada [Meilán, 2018]. Aun así, todavía es necesario seguir avanzando en esta área para poder asegurar la conservación de todo este trabajo observacional.

Resumiendo, la FCAG tiene almacenadas miles de placas espectrográficas que son vulnerables al paso del tiempo. Si se quiere que sean de utilidad es prioritaria su digitalización antes de que se corrompa la totalidad de los datos que contienen. No son pocas las instituciones que poseen colecciones de similares características. Harvard [DASCH, 2017], el observatorio de Heidelbergel [Mandel et al., 2007] o el observatorio Maria Mitchell [Mitchell, 1995] por mencionar algunas. Si se lograse una forma adecuada y eficaz para resolver el problema, la misma podría difundirse para que sea de utilidad para digitalizar patrimonios similares.

Afortunadamente, los recientes avances en el campo de Inteligencia Artificial presentan herramientas adecuadas para automatizar parcialmente la digitalización de las placas: aprovechando modelos Aprendizaje Automático para recortar sus espectros y automatizando el calculo de metadatos de cada archivo en base a procesamiento interno y consultas a bases de datos externas (como [Wenger, M. et al., 2000]).

Bajo este contexto se justifica la construcción de un software que asista con el proceso de digitalización, reduciendo el tiempo necesario para digitalizar cada placa y asistiendo en consecuencia con su conservación. Además, tener los archivos de forma digital ayudará enormemente con su difusión.

1.3. Objetivos

El objetivo de esta tesina es el desarrollo de un software que agilice el proceso de escaneado de placas astronómicas antiguas (patrimonio histórico de la UNLP), con el objetivo de recuperar la información guardada en las mismas en el menor tiempo posible.

Como objetivo específico se propone el desarrollo de un software de digitalización de placas, procesamiento automático de imágenes, y carga de metadatos, con los siguientes requerimientos:

- Automatizar el llenado de metadatos utilizando consultas a bases de datos astronómicas externas.
- Implementar un sistema inteligente que sea capaz de segmentar de forma correcta y automática cada uno de los espectros que se registran en los escaneos de placas espectrográficas recibidas.
- Implementar un servidor que integre adecuadamente las funcionalidades antes descriptas y que facilite su uso para un cliente mediante una API tipo REST.
- Implementar una aplicación cliente donde el usuario interactuará con la aplicación, y definirá el flujo del proceso de digitalización, delegando todo el procesamiento complejo al servidor antes mencionado.

1.4. Organización del documento

El documento se divide en 5 capítulos.

El capítulo 1 introduce al lector al trabajo realizado y enumera los objetivos de la tesis.

En el capítulo 2 se presenta el marco teórico astronómico: se explica el material astronómico con el que se trabajó, junto a sus principales características y se exploran las características de los principales formatos de archivos astronómicos con los que trabaja sistema desarrollado.

En el capitulo 3 se presentan los conceptos informáticos necesarios para comprender el trabajo realizado: se explican los lenguajes de programación usados, los principales frameworks con los que se trabajo y los conceptos de Inteligencia Artificial relevantes para el entendimiento de la presente tesis.

En el capítulo 4 se empieza la presentación del desarrollo realizado, recorriendo el Frontend, su estructura, sus distintas funcionalidades y el flujo de uso que se pretende del mismo.

En el capítulo 5 se continúa mostrando el desarrollo realizado, pero en este caso desde el Backend: se explican sus principales características, se muestra las distintas funcionalidades que ofrece en su API y explica detalladamente cómo resuelve sus funcionalidades mas complejas.

Por último, en el capítulo 6 se presentan las conclusiones obtenidas y se listan posibles líneas de trabajo a futuro para continuar con lo desarrollado en esta tesina.

Capítulo 2

Marco teórico (Astronomía)

Las ciencias astronómicas tratan sobre el origen, la composición, la evolución, la distancia y el movimiento de todos los cuerpos celestes y de la materia dispersa en el universo [Enríquez et al., 2005]. Son de las primeras ciencias de las que se empezó a tener registro, estimando que fueron pacientemente exploradas aún antes de la formación de las primeras civilizaciones [Oster, 2021]. Naturalmente, con el paso del tiempo sus concepciones del mundo y las técnicas que emplea fueron cambiando.

Actualmente, las tecnologías de recopilación de datos en conjunto con su bajo coste de almacenamiento, permiten cotas sin precedentes en la cantidad de datos que las instituciones dedicadas a estas ciencias tienen disponibles [Mayer-Schönberger and Cukier, 2013]. Se generan incontables datos digitales (fotos, vídeos, etc) del espacio en cada momento y los continuos desarrollos en las ciencias de computación hacen posible su procesamiento. Sin embargo, durante el siglo pasado esto no era así. En ese entonces, el recopilado de información era mas analógico, los instrumentos tendían a generar datos en formatos físicos (placas, fotografías, cuentas anotadas, etc). La cantidad de información que se recopiló en ese tiempo fue sin duda considerable pero la interpretación y puesta en valor de la misma se hallaba limitada por la cantidad de personal que podía analizarla. Se generaba mas información de la que los científicos podían procesar y muchas instituciones optaron por guardar el excedente de datos para que más adelante pueda ser aprovechada. Luego, aparecieron nuevos instrumentos y formatos digitales que presentaban mayores facilidades que los anteriormente usados por lo que muchos de los instrumentos analógicos se volvieron obsoletos. Estas placas, fotos, notas, etc que se almacenaron quedaron olvidadas ya que muchas instituciones no tenían los recursos necesarios para procesarlas. No obstante, el paso del tiempo no hizo que su registro sea menos valioso ya que muchos de estos datos son únicos debido al tiempo y lugar que fueron tomados. Los mismos hablan de las estrellas y su evolución, muestran explosiones de supernovas imposibles de predecir, permiten calcular la velocidad de estrellas muy distantes como para notar su desplazamiento en cortos periodos de tiempo, entre otras cosas [Meilan, 2020].

Pese a que es difícil trabajar con ellas su información es valiosa y así como los tiempos modernos supusieron su olvido, también traen nuevas oportunidades. Las innovaciones actuales en las ciencias informáticas nos sugieren que el procesamiento masivo de estos datos es posible, siempre y cuando los registros sean convertidos a un formato digital apto para las herramientas modernas. Si esto se hace y se aplican las técnicas adecuadas entonces su almacenamiento no habrá sido en vano, además de que también será mas fácil la difusión de sus datos.

Justamente, el trabajo aquí realizado trata esta problemática pero para comprenderlo en su totalidad es recomendable conocer ciertos aspectos de las ciencias astronómicas. Información respecto a formatos y recursos inherentes al problema, que fueron importantes al realizar el diseño del software. El presente capítulo procura hacer un breve recorrido por toda la información relevante de esta área. En la sección 2.1 se habla de los formatos en los que se encuentran o se busca que estén los datos con los que se va a trabajar y en la sección 2.2 se habla sobre los objetos de esta tesis, las placas espectrográficas, se explica que son, como es su anatomía, por que es necesario conservarlas y cual es su estado actual.

2.1. Formatos estándar

En esta sección se explican las características de los formatos de archivo FITS y TIFF, relevantes para el presente trabajo.

2.1.1. FITS

Origen

El formato FITS (*Flexible Image Transport System*) nació en 1977 a manos de la WSIPR (*Workshop on Standards for Image Pattern Recognition*) como una alternativa simple, flexible y general a los diversos formatos de imagen que usaba cada instituto u observatorio. En ese entonces, cada vez que un instituto colaboraba con otro debía encontrar una forma de convertir los archivos de su formato al formato que usase el otro y viceversa. Por eso, la aparición del formato FITS fue bien recibida como una posible solución a este problema.

Con el paso del tiempo muchos institutos fueron adoptando el formato FITS hasta el día de hoy donde es el formato predominante en el ámbito astronómico, siendo que grandes exponentes de este campo como la NASA (*National Aeronautics and Space Administration*) lo usan como archivo de distribución en la mayoría de sus operaciones. También es usado incluso fuera del ámbito astronómico, por ejemplo para la digitalización de documentos manuscritos del vaticano [Bernabé-Mónica, 2016].

Características

A diferencia de muchos formatos de imagen, el formato FITS está diseñado específicamente para datos científicos por lo que incluye muchas funciones para describir información de calibración espacial y fotométrica, así como metadatos del origen de la imagen.

Sus metadatos se almacenan en encabezados legibles por humanos (formato ASCII), para que cualquier usuario interesado los pueda analizar sin mayores complicaciones. Los encabezados son pares -*Clave,Valor*- y cada archivo debe de tener como mínimo un encabezado. De estos se obtiene información como tamaño, origen, coordenadas, formato de datos binarios, comentarios, historial de datos y cualquier otra cosa que se quiera especificar; muchas palabras claves están reservadas para el uso interno, pero las demás se pueden usar a discreción del escritor.

Estructura

Los archivos FITS consisten en un encabezado primario seguido de uno o mas encabezados de extensión, cada de uno de los cuales posee un conjunto de datos que le corresponde, tal como se muestra en figura 2.1.

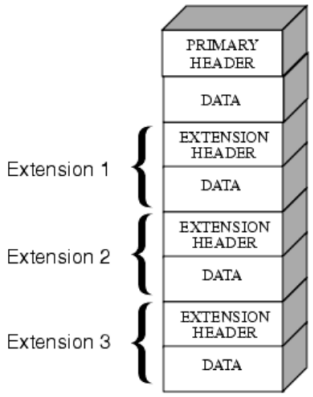

Figura 2.1: Estructura típica de un archivo FITS.

Los archivos no necesariamente tienen que almacenar imágenes, el formato FITS soporta otros tipos de estructuras, como espectros o cubos de datos, pero para este trabajo particular interesa su forma de uso mas común, la cual consiste en adjuntarle una única imagen en conjunto de varios metadatos.

Medios de uso

El formato es bastante desconocido fuera del ámbito científico, por lo que sus imágenes no se pueden abrir con software de edición de imágenes convencional.

Algunos software populares de procesamiento de imágenes pueden leer archivos FITS simples, como GIMP o Photoshop, pero su visualización es limitada ya que no interpretan las tablas y datos complejos de los mismos. Para poder analizar adecuadamente un archivo FITS se requiere usar software especializado, como ImageJ [Schroeder et al., 2020] o SAOImage DS9 [Joye and Mandel, 2005].

El formato es soportado por bibliotecas estándar de varios de los principales lenguajes de programación que se usan en entornos científicos, como: C, Fortran, Java, Perl, Python e IDL. Además, la Oficina de soporte de NASA, GSFC FITS o también conocida como "The FITS Support Office", mantiene una lista de bibliotecas y plataformas que admiten el formato FITS[1].

2.1.2. TIFF

Origen

El formato TIFF (*Tagged Image File Format*) fue desarrollado en 1986 por Microsoft, la ya desaparecida Aldus Corporation y HP (Hewlett Packard). Fue concebido como un formato para la estandarización del intercambio de imágenes. Si bien no lo logró del todo, hoy en día es muy popular para la generación de fotografías de alta calidad, digitalizaciones de alta resolución o como archivo contenedor para serializar varios archivos de menor tamaño (como JPEG).

Características

Un archivo TIFF admite imágenes codificadas en escala de grises, RBG, CMYK y espacio de color LAB, permitiendo una profundidad de color de hasta 16 bits por canal de color.

Uno de sus principales puntos fuertes es que a diferencia de otros formatos de archivo como JPG, la compresión y descompresión de un formato TIFF habitualmente no tiene pérdidas, pudiendo reducir el tamaño del archivo sin afectar negativamente la calidad original, conservando los detalles y la profundidad de color de las imágenes originales, lo que resulta idóneo para fotografías profesionales o imágenes de gran calidad. Sin embargo, esto tiene un lado negativo, ya que su gran calidad es a costa de que los mismos ocupen una considerable cantidad de espacio, por lo que en entornos donde se aprecia la velocidad de carga puede ser mejor usar formatos mas ligeros como JPEG.

Analizando sus siglas podemos notar que TIFF corresponde a "*Tagged Image File Format*", o sea "Formato de archivo de imagen etiquetado" lo cual hace referencia a que un archivo con esta extensión admite que sus datos sean etiquetados. De esta forma, se le permite a los usuarios agregar etiquetas para contener *headers* o metadatos.

[1]The FITS Support Office: https://fits.gsfc.nasa.gov/

Medios de uso

Los archivos TIFF están ampliamente difundidos, por lo que los principales software de edición de imágenes suelen soportar su visualización y edición. Algunos ejemplos de software que lo soportan son Photoshop o Krita.

2.2. Placas Espectrográficas

Una placa espectrográfica (o espectroscópica) consiste en un pequeño rectángulo de vidrio, que en la superficie de una de sus caras tiene una emulsión fotosensible, sobre la que se aprecian espectros de luz, cada cual corresponde a un tiempo y región astronómica determinados. En la figura 2.2 se puede ver un ejemplo de este tipo de placas.

El proceso para grabar un espectro en una placa es relativamente simple, la idea es exponer la emulsión de la placa a los fotones del objeto que interesa estudiar, no sin antes dispersar la luz mediante un prisma o una red de difracción. Al hacerlo el espectro resultante de dispersar la luz queda grabado en la superficie de la emulsión fotosensible de la placa. Si la luz no se dispersara, en la placa quedaría registrada una imagen de la región estudiada, obteniendo lo que se conoce como placa fotográfica en vez de una placa espectrográfica. En general, mientras mas intensa es la señal más se oscurece la emulsión de la placa.

Al momento de grabar un espectro también se usan las denominadas *lámparas de comparación*. Estas están compuestas de pequeños cilindros, donde cada cual contiene una mezcla de gases particular y se corresponde a un elemento químico determinado. Al hacer reaccionar la emulsión se colocan dos lámparas de comparación a cada lado, obteniendo al final de la reacción el espectro del objeto que se busca estudiar, junto a dos espectros parecidos a un código de barras a cada uno de sus lados, los cuales surgen de las lámparas de comparación colocadas. Los patrones que se graban en estas barras extra ayudan a diferenciar qué sección del espectro base se corresponde con cada elemento químico. En la figura 2.5 se pueden ver 4 espectros cada uno con sus respectivas lámparas de comparación.

Además del segmento de vidrio con el espectro y sus respectivas lámparas de comparación, también se suele tener adjunto a cada placa varios datos anotados a mano, ya sea en el papel donde se guarda la placa, o a veces en libros de registros que se llenaron conforme se realizaban las distintas observaciones.

En la figura 2.2 se puede ver como queda una placa luego de reaccionar. Notar que la misma tiene 3 espectros grabados en su superficie, esto es así debido a que en su momento se optaba por grabar múltiples espectros en cada placa con el fin de ahorrar materiales. Esto no traía consigo desventajas evidentes por lo que de fué una práctica bastante común.

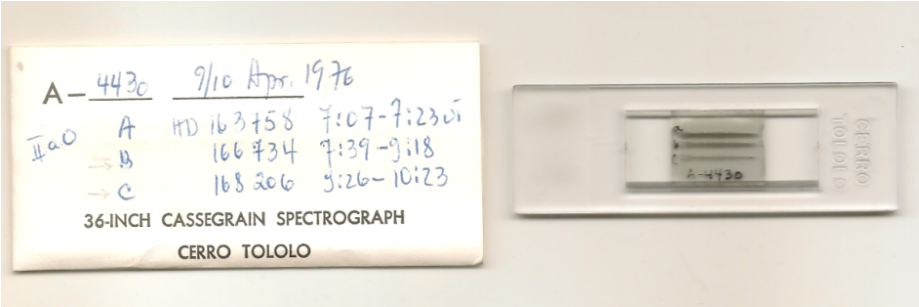

Figura 2.2: Placa espectrográfica de vidrio con tres espectrogramas, junto a sus correspondientes anotaciones en formato papel.

2.2.1. Información de un espectrograma

Un espectro tiene información respecto a la cantidad de energía por longitud de onda que se percibe en el objeto de estudio. Cada sección del espectro se corresponde a una longitud de onda determinada. El espectro de cada elemento químico tiene un patrón único (como una huella digital) y estos patrones se pueden reconocer en los espectros de interés haciendo uso de las antes mencionadas *lámparas de comparación*. De cada lámpara se sabe el material que le corresponde por lo que usando su patrón como referencia se puede distinguir a que elemento o conjunto de elementos corresponde cada sección del espectro analizado. De esta forma se puede conocer que elementos químicos (hierro, bohrio, etc) son los que forman los objetos que se estén estudiando.

2.2.2. Patrimonio histórico único y fuente de nuevos descubrimientos

La FCAG tiene en su poder una colección de mas de 15000 espectros impresos en placas espectrográficas. Las placas datan de entre las décadas de 1900-1980 y fueron tomadas con instrumentos instalados en el Observatorio Astronómico de La Plata (OALP), el Observatorio de Bosque Alegre de Córdoba (OAC) y Cerro Tololo (CTIO) en Chile. Su importancia radica en que constituyen un patrimonio histórico único, son testigos de las distintas campañas y misiones internacionales en las que participó nuestro país, tales como las campañas de seguimiento de eclipses total de sol en pos de confirmar la Teoría General de la Relatividad (1914-1919). Además, la información que contiene este tipo de registros ha demostrado ser de utilidad en numerosos trabajos que los usaron para respaldar sus conclusiones [Paolantonio, 2021a, Paolantonio, 2021b, Rieznik, 2013, Bernaola, 2004] o como base para realizar nuevos descubrimientos [Wertz et al., 2017, Muminov et al., 2017, Walborn et al., 2017].

Sin embargo, el método de recopilación de información astronómica que se usó para su concepción actualmente está en desuso. Los equipos que se usaban para el procesamiento de este tipo de placas (como los densitómetros) ya no

están operativos. El formato analógico en que los datos observacionales están disponibles tampoco es útil bajo criterios modernos. Por último, estas están cada vez mas deterioradas por el paso del tiempo.

Por estas razones, en el año 2000 la UAI (Unión Astronómica Internacional) emitió la resolución *Nro. B3 'Safeguarding the information in photographic plates'* [UAI, 2000], donde solicita que se tomen medidas para conservar los datos históricos de este tipo, a fin de evitar que estos se pierdan para las futuras generaciones de astrónomos. Posteriormente en el año 2019 la FCAG, bajo la dirección de la Dra. Lydia Cidale y el Dr. Roberto Gamen, gestó el proyecto ReTrhOH (proyecto de Recuperación del Trabajo Observacional Histórico)[ReTrhOH, 2019]. En este proyecto se comenzó a realizar un proceso de digitalización sobre la ya mencionada colección de placas espectrográficas.

Sin embargo, el proceso manual de digitalización es lento y se debe dedicar una considerable cantidad de tiempo a cada placa. Por ejemplo, si se dispusiera de una persona que se dedique a este problema con una jornada diaria de ocho horas y con una tasa de procesamiento optimista de una hora por registro, entonces se podría completar el procesamiento de las placas en aproximadamente ocho años. Considerando estos tiempos, es necesario reducir el tiempo de procesamiento requerido por cada placa, en pos de lograr la digitalización total de las placas en el menor tiempo posible.

2.2.3. Escaneo de placas físicas

La FCAG ha tenido avances respecto a la posibilidad de trabajar con estas placas por medio de medios digitales y ha desarrollado un método de escaneo que mantiene toda la información correspondiente a las placas físicas de las que se originan [Meilán, 2018].

El escaneo se realiza mediante un escáner NIKON 9000ED, con ligeras modificaciones, como el que se muestra en la figura 2.3. Las modificaciones son únicamente relacionadas al carril donde se ingresa el material a escanear, ya que las placas son pequeñas respecto al carril. Por ello necesitan una máscara contenedora que la sujete firmemente y que haga contacto con los bordes del carril, de tal forma que evite el deslizamiento de la placa durante el escaneado. Además, las máscaras se confeccionan con materiales oscuros y opacos con el fin de evitar cualquier tipo de reflexión de luz que pudiesen perjudicar el escaneo resultante. En la figura 2.4 se puede ver un ejemplo de las máscaras usadas.

Se pueden encontrar más detalles respecto a todo el proceso de escaneado y la especificación detallada de las máscaras en el trabajo de tesis en que los mismos fueron desarrollados *Recuperación del patrimonio observacional histórico* de Natalia Soledad Meilán [Meilán, 2018].

Figura 2.3: Escáner NIKON 9000ED modificado para la digitalización de placas espectrográficas.

Figura 2.4: Placa espectrográfica con su correspondiente máscara.

Los archivos resultantes del escaneo con este método están en formato TIFF, el mismo es adecuado para escaneos ya que posee una amplia calidad de imagen y no pierde información con su compresión, pero como se mencionó en secciones anteriores el formato estándar para este tipo de archivos astronómicos es el FITS por lo que para su posterior almacenamiento es necesario convertirlos de un formato a otro. El trabajo de tesis antes mencionado [Meilán, 2018] ya presenta un posible método de conversión mediante la consola de comandos. Sin embargo es un método manual y que carece de automatización, la cual sería conveniente, considerando la cantidad de placas que se deben procesar. Además, la automatización puede reducir los errores estocásticos provocados por la definición e inserción manual de datos. En la figura 2.5 se puede ver el escaneo resultante de una placa espectrográfica con 4 espectros en su superficie.

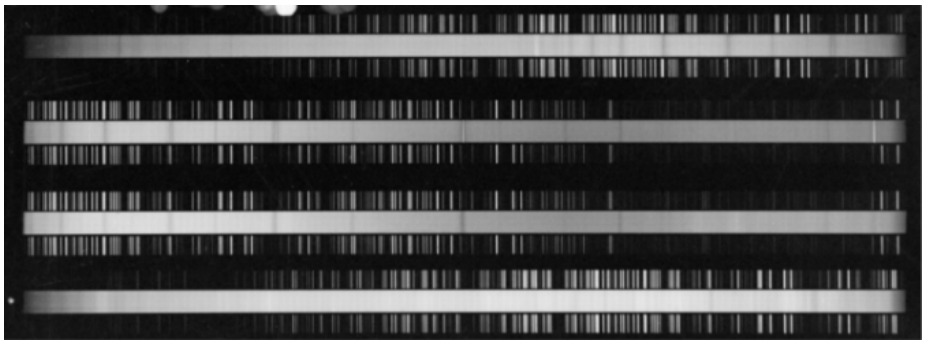

Figura 2.5: Escaneo de una placa espectrográfica con 4 espectros distintos, cada uno con sus correspondientes lámparas de comparación.

2.2.4. Situación de las placas

Ya datan cuatro décadas desde que las placas fueron tomadas. Desde entonces la gran mayoría se han mantenido almacenadas para su conservación. Muchas de las placas siguen aptas para ser analizadas, pero hay algunas que debido a múltiples circunstancias se considera que están defectuosas, parcial o totalmente. Para ello, se diferencian 3 razones principales:

- Almacenamiento prolongado: El paso del tiempo ha dejado marca en muchas placas, principalmente por lo volubles que son sus materiales y por la calidad de los métodos de almacenamiento usados. En general, mientras mejores sean las condiciones de almacenamiento menos sufrirá la placa por el paso del tiempo, pero en la práctica, es común que en una colección se encuentren varias placas que no hayan logrado mantenerse impolutas. Los defectos que se encontraron relacionados a esto son suciedad, rayones o manchas por humedad.

- Problemas al realizar la observación: Se trata de posibles circunstancias o eventualidades que pudieran haber ocurrido al momento de preparar la placa y al hacer que el reactivo reaccione a los fotones del espacio. Defectos que se encontraron relacionados a esto fueron rayones o que los espectros se hayan grabado en curvas (se esperan que los mismos se graben como una línea recta).

- Falta de elementos del espectro: Es difícil decir con certeza por qué ocurre, pero en algunas ocasiones el espectro carece de sus lámparas de comparación. También existe la situación inversa, donde se tienen las lámparas de comparación, pero, sin el espectro que les corresponde. En estas situaciones los espectros y/o lámparas deben ser descartados, ya que se requieren ambos elementos para analizar los datos de forma apropiada.

En la figura 2.6 se pueden ver ejemplos de los defectos antes descriptos.

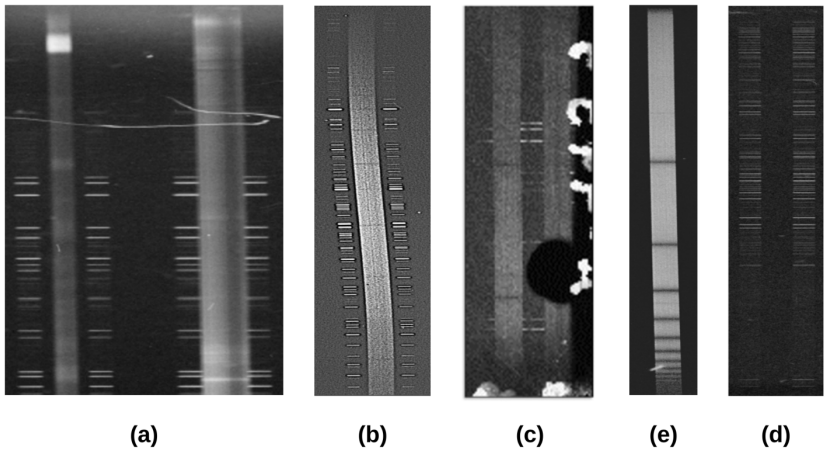

Figura 2.6: Muestras de posibles defectos en las placas. (a) Rayones. (b) Grabado en forma curva. (c) Mancha de suciedad y/o humedad. (e) Falta de lámparas de comparación. (d) Lámparas de comparación sin su respectivo espectro.

2.2.5. Análisis post conversión

Una vez los espectros están en un formato apto (FITS), estos pueden ser dispuestos en tareas de análisis y clasificación. Usualmente consisten en el uso de software especializado como *IRAF* para generar funciones que describen la información de los espectros (aunque en la actualidad debido a la falta de mantenimiento de IRAF esta habiendo una lenta migración al uso de python para resolver esta funcionalidad). Estas funciones son calibradas y normalizadas según sus lámparas de comparación para así generar otras funciones que aíslan la información de interés. Información que es usada para realizar tareas tales como análisis morfológicos de los espectros o para realizar clasificación espectral en los mismos [Meilán, 2018].

Todo este análisis es realizado una vez el espectro ya está digitalizado, lo que suele implicar que su conservación no está comprometida, por lo menos no al mismo punto que cuando estaban en un formato físico. Además, el manejo digital de los espectros trae otras ventajas, como la rápida generación de copias o las facilidades de difusión que los medios digitales proveen.

Esta información es proveída para tener conciencia respecto a que es lo que se hace con los espectros una vez son digitalizados, pero no se lo confronta en este trabajo, por lo que un mayor detalle sobre este proceso de análisis y clasificación es innecesario.

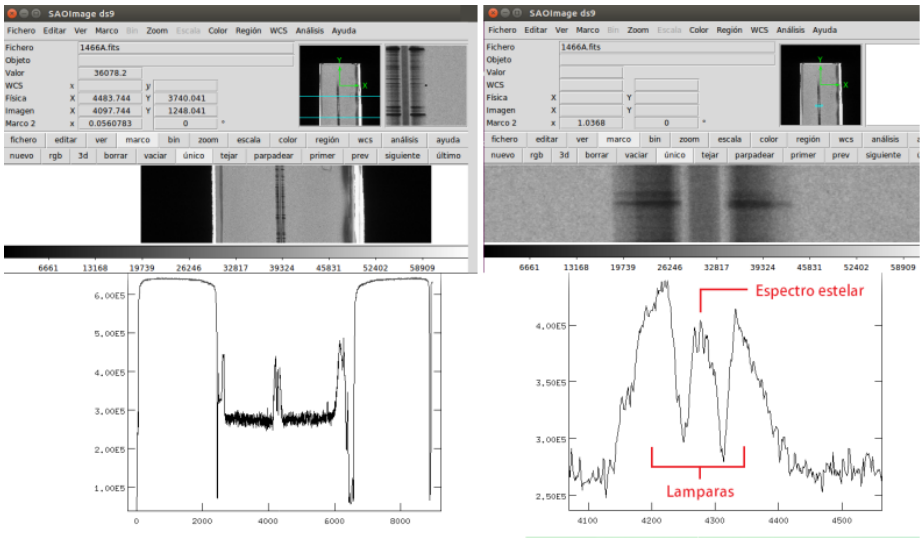

Figura 2.7: Conversión con SaoImage DS9 de una imagen de un espectro a una función [Meilán, 2018].

2.2.6. Software de digitalización

Actualmente no hay software que ayuden en especifico con el trabajo de digitalización de placas espectrográficas. No obstante, existe software que asiste en trabajos de digitalización similares.

Un ejemplo es *PyPlate* [Taavi et al., 2014] el cual es un paquete python que se usa principalmente para realizar tareas relacionadas al flujo de trabajo del procesamiento de placas fotográficas. De hecho, ya se lo ha usado para la digitalización de varias placas publicadas en la base de datos APPLAUSE [Detlef et al., 2014].

De base, el problema para el que fue construido es similar, teniendo abundantes cantidades de placas que es necesario digitalizar para que puedan ser procesadas adecuadamente. Sin embargo, no se enfoca sobre el mismo tipo de placas. *PyPlate* fue diseñado para trabajar sobre placas *fotográficas*. En contraste, el presente trabajo se centra en la digitalización de placas *espectrográficas*.

Pese a la similitud de sus nombres, estos dos tipos de placas son esencialmente distintos. Una placa espectrográfica se consiste en la reacción de un reactivo químico a fotones dispersos de una sección astronómica concreta, similar a como un prisma descompone la luz que recibe en varios colores. Mientras que una placa fotográfica, es como una fotografía siendo que se centra en capturar las formas, las posiciones y el brillo de los objetos astronómicos, similar a lo que se percibe al verlos desde un telescopio. En la figura 2.8 se puede ver una comparación entre los 2 tipos de placas.

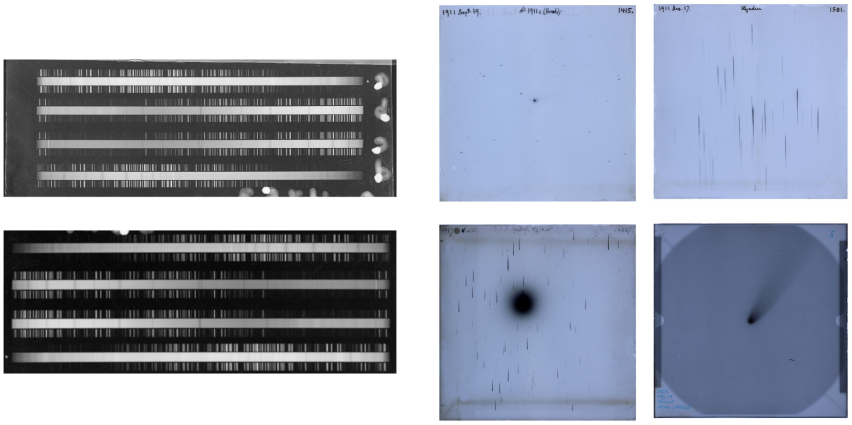

Figura 2.8: Ejemplos de distintos tipos de placas. (a) Placas espectrográficas. (b) Placas fotográficas [Meilán, 2018].

PyPlate principalmente provee utilidades para leer metadatos, realizar cálculos derivados de estos, guardarlos en archivos FITS o en bases de datos MySQL y ayuda en la integración con otros softwares como *SExtractor* para la extracción de fuentes de imágenes o *SCAMP* para calibración astrométrica. Pero todo está centrado en placas fotográficas, por lo que muchas de estas funciones no son útiles para el procesamiento de placas espectrográficas. Además, *PyPlate* carece de interfaz gráfica, al ser un paquete de python, por lo que es difícil hacer uso del mismo por parte de usuarios sin conocimientos de programación.

Capítulo 3

Marco teórico (Informática)

En este capítulo se explican distintos conceptos y tecnologías que son de interés para el trabajo realizado. A lo largo de las siguientes secciones se habla de los lenguajes y frameworks que se usaron y se realiza una breve introducción a los conceptos de Inteligencia Artificial relevantes para este trabajo.

3.1. Lenguajes de programación

A continuación se introduce brevemente los lenguajes de programación que se usaron durante el esta tesis.

3.1.1. Python

Python es un lenguaje de programación de propósito general, el mismo puede usarse para una gran variedad de proyectos: aplicaciones pequeñas, grandes, servidores, entre otros. No obstante se da principalmente en los ámbitos de desarrollo web y análisis de datos.

Es interpretado, por lo que su eficiencia en ejecución es peor comparada a la de otros lenguajes compilados como Java. Sin embargo, esto no ha impedido que diversas organizaciones como la NASA, Google o Youtube usasen el lenguaje en sus propios proyectos. Python ofrece varias ventajas, tales como:

- Está disponible en todas las principales plataformas (Windows, Linux, etc).
- Es de código abierto.
- Es de descarga y uso gratuito.
- Tiene una sintaxis única que se centra en la legibilidad, por lo que su código es fácil de leer y traducir, lo que en consecuencia también reduce el tiempo que requiere un nuevo programador para comprender un programa, o el propio lenguaje.

- Está ampliamente difundido, por lo que es fácil encontrar online soluciones a problemas comunes, además de una basta cantidad de tutoriales que explican desde los conceptos mas básicos, hasta aspectos más especializados sobre paquetes o tecnologías concretas.

En definitiva, es un lenguaje de programación relativamente fácil de aprender, altamente difundido, con una gran comunidad y todo lo que requiere para su uso está disponible para todos de forma gratuita.

Encapsulamiento vía entornos virtuales

La administración de librerías en python se resuelve mediante el comando *pip*. Cuando una librería es instalada cualquier proyecto que se ejecute en el mismo equipo también la tendrá disponible. Esta mecánica es poco deseable ya que es regular que distintos proyectos manejen distintos conjuntos de librerías y que estos sean incompatibles entre sí. Como solución Python ofrece la herramienta *virtualenv*, que permite hacer uso de entornos virtuales cada hace uso de su propio conjunto de librerías, pudiendo así definir las librerías de distintos proyectos de forma independiente.

3.1.2. HTML

HTML (HyperText Markup Language), es como su nombre indica un lenguaje de marcas. Es el estándar actual para la definición del contenido que muestra una página web. Mediante algunos complementos permite mostrar gran variedad de contenidos y se está actualizando continuamente, por lo que con el paso del tiempo también va ampliando sus capacidades.

El estándar HTML está a cargo de la organización W3C (World Wide Web Consortium), la que decide qué cambios se incorporan al estándar. Para que los cambios se hagan efectivos los propios navegadores tienen que incorporarlos pero el hacerlo queda a discreción de los responsables de cada navegador. Puede ocurrir que la apariencia de una misma página varíe de un navegador a otro, dependiendo de cómo estos implementan el estándar.

El cambio mas relevante que tuvo en los últimos años fue en 2014 cuando se publicó HTMLv5, la cual es una gran actualización del estándar que se tenía hasta el momento. HTML5 mantiene muchas características de las versiones anteriores que funcionaban bien, pero simplifica otras que eran innecesariamente complicadas. También incorporan otras nuevas que permiten explotar mejor las capacidades de los navegadores actuales, como el soporte de forma nativa de audio y vídeo.

Estructura

Como ya se dijo el lenguaje se basa en el uso de marcas que indican determinadas características, un ejemplo sencillo de esto puede ser la etiqueta $< b >$ la cual permite destacar un texto concreto, por ejemplo si se escribe $< b > Hola$

mundo < */b* > en una archivo HTML, al visualizarlo desde un navegador entonces aparece "Hola mundo". A las etiquetas se les pueden definir atributos o también se pueden usar otras etiquetas como < *div* >, < *head* > o < *body* > para definir la estructura semántica del documento.

En general para páginas mas completas se requiere complementar HTML con código CSS y Javascript, ya que de forma individual HTML no ofrece la flexibilidad que se requiere para programas este tipo de páginas. Aunque en HTMLv5 se ampliaron las capacidades del lenguaje en múltiples aspectos, el uso de CSS y Javascript suele ser necesario en la mayoría de proyectos serios.

3.1.3. Javascript

Al igual que Python, Javascript (JS) es un lenguaje de programación interpretado y de tipado dinámico. El propio cliente por medio de los navegadores es el que actúa de intérprete (aunque puede ser ejecutado a nivel de servidor a través de ciertos softwares como Node.js. También está altamente difundido por lo que es sencillo encontrar tutoriales del lenguaje, tanto de aspectos básicos como otros mas avanzados.

Permite incorporar dinamismo y reactividad a las páginas web, como la posibilidad de ejecutar efectos, animaciones o eventos al presionar un botón. En general, en un código HTML cualquier operación que posea cierto grado de complejidad suele ser definida como una función javascript, y meramente es referenciada en la parte del código HTML donde se quiera ejecutar. La mayoría de páginas que hacen algo más que ser un conjunto de elementos estáticos (que se actualizan, son reactivas, tienen mapas interactivos) tienen código javascript incorporado. También es estrictamente necesario en caso de que se quiera que la página interactúe con algún tipo de API. Como en el presente trabajo.

3.1.4. CSS

CSS (Cascading Style Sheets) son secuencias de código que definen las características visuales de las estructuras especificadas en el código HTML. Usualmente se especifica el nombre de una etiqueta, una clase o un identificador y se le definen una serie de valores para sus atributos, los cuales definirán cómo es que se ve en el propio navegador. Se puede usar para ajustes pequeños como cambiarle el color a la letra de los párrafos o para cosas mas complejas como animar un botón al presionarlo.

Puede escribirse dentro del propio HTML que se quiera usar por medio de las etiquetas < *style* > o importando el estilo desde un archivo aparte con extensión *css*.

3.2. Frameworks

Actualmente en el desarrollo moderno de aplicaciones web se utilizan distintos frameworks, los cuales son herramientas, paquetes o librerías que proveen

utilidades y/o funciones que permiten al programador abstraerse de las principales complicaciones que implica el uso de una tecnología concreta. Suelen proveer cierto nivel de estructura al proyecto y en el caso de los frameworks mas conocidos es común encontrarse vastas comunidades que pueden ayudar en aquellos lugares en los que la documentación no responde todas las dudas. Es muy común que se los use para la construcción de páginas web dinámicas y suelen estar asociados a un lenguaje de programación determinado, como Ruby, PHP o Python.

A lo largo del resto de la sección se introducen varios de los frameworks usados en este proyecto.

3.2.1. Flask

Flask es un micro-framework, que tiene como objetivo la creación de Aplicaciones Web mediante el lenguaje de programación Python. Simplifica en gran medida la creación de aplicaciones Web y permite desplegar de forma sencilla varios *endpoits* con los que los usuarios pueden interactuar.

La palabra "micro" en su definición no implica que solo sea útil en proyectos pequeños o aplicaciones de requisitos mínimos. En realidad se refiere a que dispone todo lo necesario para armar una aplicación web común, pero si se llegase a necesitar alguna funcionalidad extra entonces se pueden intentar instalar desde el conjunto de extensiones complementarias que Flask y su comunidad ofrecen. De esta forma las funcionalidades se instalan según se requieren, evitando con ello ocupar espacio en funcionalidades innecesarias. Además, el propio desarrollador sólo necesita aprender a usar las extensiones que el proyecto requiera, ya que las demás extensiones no podrán afectar al proyecto de cualquier forma.

En la práctica, Flask se puede instalar con la herramienta *pip*, similar a como se instalan otros paquetes de Python, las extensiones como Flask-Mail usualmente son instaladas de de forma separada, también con el comando *pip*. Del lado del código Python, basta con realizar los *imports* adecuados donde se quiera aprovechar cada funcionalidad.

En la figura 3.1 se puede ver un ejemplo del código correspondiente a una pequeña aplicación web armada con Python y Flask. Esta ofrece varios *endpoints* que devuelven código HTML, todos accesibles mediante el método GET. Tanto el formato de respuesta como el método HTTP que usa cada *endpoint* es configurable con las facilidades que proporciona Flask.

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return '<h1>Hello, World!</h1>'

@app.route('/articulos/')
def articulos():
    return '<h1>Lista de articulos</h1>'

@app.route('/acercade')
def acercade():
    return '<h1>Página acerca de...<h1>'

if __name__ == '__main__':
    app.run('0.0.0.0',5000, debug=True)
```

Figura 3.1: Código de una aplicación web básica construida con Python y Flask.

3.2.2. Node.js

Node.js es un entorno, creado por los propios desarrolladores de JavaScript, que permite ejecutar código JavaScript del lado del servidor (antiguamente se ejecutaba en solo en el navegador del lado del cliente). Para la ejecución del código utiliza el motor V8 de Chrome, que convierte el código JavaScript a código máquina en tiempo real.

El entorno es controlado por eventos, hay un único subproceso que va procesando los eventos a medida que estos llegan (por ejemplo solicitudes). De esta forma puede manejar una gran cantidad de conexiones simultáneas sin caer en la sobrecarga usual que tienen otros frameworks similares, los cuales generan un nuevo subproceso por cada conexión que establecen. Además contempla la posibilidad de trabajar con otros lenguajes interpretados como Python. Es una de las opciones mas difundidas a día de hoy para la construcción de servidores web.

3.2.3. Svelte

Svelte es un framework de frontend basado en javaScript publicado en el año 2016 para el diseño de interfaces web interactivas. Proporciona el marco básico para el desarrollo de aplicaciones, además de funciones recurrentes en forma de bibliotecas.

No es la única alternativa para el armado del Frontend, es relativamente nuevo y ya hay otros Frameworks populares que resuelven este tipo de problemas (React, VueJs, Angular, etc). No obstante, Svelte tiene una serie de características que vale la pena considerar:

- Fácil de aprender: los requisitos previos para hacer un uso adecuado del framework no son particularmente exigentes. Para aplicaciones básicas basta aprender algunos conceptos básicos y tener cierta familiaridad con los lenguajes JavaScript, CSS y HTML. Para funcionalidades complejas se puede buscar guía en la documentación y/o en foros de la comunidad. No obstante, Svelte todavía es muy nuevo por lo que su comunidad no es tan amplia como la de sus principales competidores y aun le falta desarrollo en las funcionalidades mas complejas.

- Aplicaciones pequeñas: es habitual que en los frameworks convencionales no se optimice el código JavaScript respecto a la velocidad en la que se ejecuta en el navegador. Los frameworks suelen incluir en el JavaScript todas las funciones relacionadas al mismo, muchas veces también las que no se utilizan. Esto provoca que se termine inflando el código de las aplicaciones, haciendo que estas consuman mas recursos de la cuenta, notando las pérdidas principalmente en el uso de memoria y el rendimiento de la aplicación. Un dicho popular respecto a esto es que estos frameworks ayudan a estructurar las ideas pero no el código. En contrate, Svelte empaqueta sus componentes en un fichero Javascript comprimido, con el código JavaScript necesario para la aplicación. Por lo que las aplicaciones creadas con Svelte son significativamente más pequeñas que las de sus principales competidores, sin perder eficiencia por ello.

- Proceso previo de compilación: Svelte tiene un paso previo de compilación que hace necesario compilar el código de la aplicacion antes de entregarlo al navegador. A priori esto podría parecer una desventaja, pero la presencia de este paso evita el uso del sistema *Virtual DOM* (VDOM) usado por otros Frameworks como React. El VDOM sirve entre otras cosas para que los frameworks sepan ante un evento cuales son los cambios mínimos que deben realizar para actualizar el estado de una página, evitando asi recargarla en su totalidad. Pero este trabajo que realiza el VDOM tiene su coste, y es peor cuanto más compleja es una aplicación. Svelte, por otro lado, envuelve los cambios de estado y propiedades en métodos que podrían directamente actualizan el DOM, evitando así cualquier costo extra proveniente del armado y mantenimiento del VDOM.

Mas allá de sus características insignia vale la pena hablar respecto a cómo Svelte estructura su código. Las aplicaciones Svelte constan de varios archivos con extensión *svelte*. A cada uno de estos archivos se los llama componentes. Cada componente sirve para definir un elemento de la interfaz web, dentro del archivo de cada componente la sintaxis es similar a la de una página web cualquiera. Un componente puede usar mediante etiquetas HTML a otros componentes, también les puede pasar valores para que estos usen en forma de atributos. El uso de componentes se puede ir extendiendo formando una jerarquía de componentes y en la cima de la misma estará el componente principal el cual representa la propia página web. Esta organización por componentes también sirve para evitar en gran medida la duplicación de código.

Respecto a la reactividad, Svelte la soporta haciendo uso de variables declaradas en el código JavaScript, cuando cambia el valor de una variable, por la interacción del usuario o como efecto de la ejecución de una función, Svelte determina que hubo un cambio de estado por lo que realiza un re-renderizado del componente actual. En la figura 3.2 se puede ver un ejemplo de cómo el cambio en una variable puede verse reflejado en la página. Además, el valor de una variable pueden ser enlazadas a los atributos de un input u otros elementos HTML, por lo que se puede hacer que el usuario los modifique de forma relativamente sencilla.

a)

App.svelte

```
<script>
  let value = 'mundo';
  // a los 2 segundos,
  // cambia el texto
  setTimeout(function () {
    value = 'Argentina'
  }, 2000)
</script>

<h1>Hola {value}!</h1>

<style>
  h1 {
    font-weight: 100;
    text-transform: uppercase;
  }
</style>
```

b)

c)

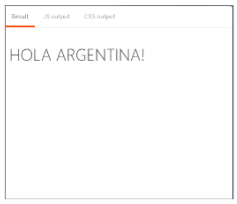

Figura 3.2: Ejemplo de reactividad con Svelte mediante la modificación automática de una variable luego de 2 segundos. a) Código de *App.svelte*. b) Página vista desde el navegador antes de que pasen 2 segundos. c) Página vista desde el navegador luego de que pasen 2 segundos.

3.3. Inteligencia artificial y detección de objetos

La IA (Inteligencia Artificial) es un campo de las ciencias de la computación que busca comprender los principios del comportamiento inteligente. Su motivación principal es que si se comprenden los principios del comportamiento inteligente para una determinada tarea entonces esta tarea tiene el potencial de ser explicada como un modelo matemático y por ende traducida en forma de un algoritmo que pueda ejecutarse en un sistema computacional. Aunque ahora parezca lejano, las propias calculadoras en su momento fueron dispositivos en el tope de la innovación tecnológica, considerando la capacidad que tenían de realizar cuantas complejas en milésimas del tiempo que le tomaba a un empleado regular. Hasta cierto punto se puede decir que las calculadoras son inteligentes, pues estas incorporan toda la lógica necesaria para resolver tareas de cálculo de forma similar a la humana, usualmente con un tiempo mejor. No obstante, en la actualidad los requisitos para considerar que un sistema hace uso de Inteligencia Artificial son mas difusos. Para algunos, cualquier dispositivo que pueda resolver una tarea de forma similar o mejor a como lo haría un humano se puede catalogar como inteligente. Para otros, el funcionamiento de una simple calculadora que interpreta *inputs* y decide de forma programada que debe hacer no es suficiente para catalogarla con esta etiqueta. En general, para decir que un algoritmo es inteligente este debe ser capaz de interpretar los datos de su entorno y en base a estos, definir los criterios que seguirá para resolver el problema. Osea que, no solo tiene que resolver el problema, sino que tiene que aprender en base a información externa el *cómo* hacerlo.

En esta última concepción de IA es en la que se centra lo que se conoce

como AA (Aprendizaje Automático). El AA busca que en lugar de que una computadora actúe en función de instrucciones explicitadas previamente por un humano, esta aprenda la forma correcta de realizar una tarea en base a un conjunto de datos. Existen varios métodos y modelos con los que se pueden obtener programas con estas características pero el que mas interesa conocer para los fines de este trabajo es el AAP (Aprendizaje Automático Profundo).

El AAP consiste en la creación y entrenamiento de algoritmos que procesan los datos de forma similar a la humana, basándose para ello en implementaciones de modelos matemáticos que imitan el funcionamiento del cerebro humano, implementaciones mejor conocidas como RNA (Red Neuronal Artificial). Se puede pensar en las RNA como un conjunto de nodos agrupados en distintas capas interconectadas. La primera y última capa son denominadas capa de entrada y salida respectivamente, un nombre bastante acertado si consideramos que la capa de entrada recibe los datos iniciales con los que el modelo tiene que dar una respuesta y la capa de salida devuelve la respuesta como tal. Cada nodo de una capa es identificado como neurona y así como una neurona biológica su funcionamiento se basa en la premisa de recibir una señal de entrada, capaz transformarla de alguna forma y propagarla a todas las neuronas con las que esté conectada y que sean alcanzables en base a la señal transmitida. Esta premisa es claramente visible en el modelo de RNA, ya que cada nodo posee un peso y un umbral asociado. El peso actúa como un modificador con el que cada nodo realiza operaciones matemáticas según los valores recibidos como input y el umbral determina que tan grande debe ser el peso para transmitirse al próximo nodo. Si la información que recibe un nodo está por encima de un determinado umbral entonces se activa y retransmite la información a los nodos de la siguiente capa, los cuales seguirán la misma regla. Se puede ver el diagrama correspondiente a una RNA en la figura 3.3

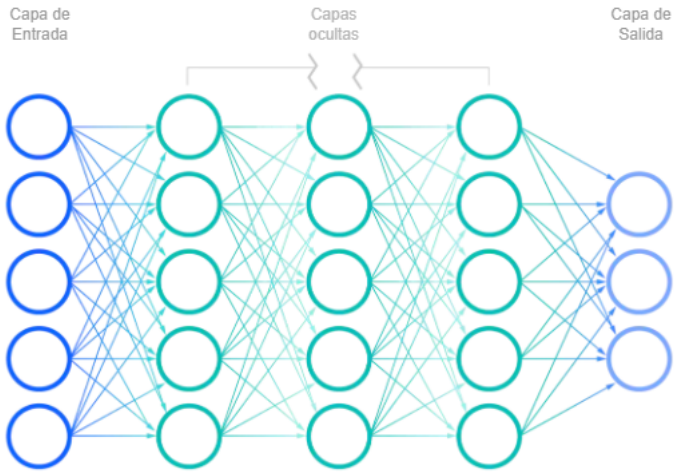

Figura 3.3: Representación gráfica de una RNA [IBMCloudEducation, 2020].

Los pesos y umbrales son inicializados de forma aleatoria y se realizan varias iteraciones de entrenamiento conforme un conjunto de datos de entrenamiento de los que se conoce la salida correcta que el modelo debería dar. La RNA procesa cada dato de entrada y da su resultado, usualmente equivocándose con los primeros valores debido a los valores de inicialización pero con la información de cada ejemplo y el resultado obtenido se puede hacer uso de funciones de error como ECM (Error Cuadrático Medio) para obtener estimaciones numéricas de qué tan errónea fue la predicción realizada por el modelo. Con estas estimaciones poco a poco se realizan correcciones sobre los pesos y umbrales configurados en cada nodo. Todos los ajustes orientados al objetivo de minimizar el ECM del modelo. Conforme se minimiza el ECM el la RNA tiende a realizar mejores predicciones, aunque todo depende de la calidad de los datos que se haya dispuesto inicialmente. Si los datos que se usaron para entrenar la RNA tenían incorrectamente asignados sus resultado esperado, si se tenían muy pocos datos o si el conjunto de datos no era realmente representativo de los datos con los que se planea usar la RNA entonces es probable que no se pueda obtener un RNA lo suficientemente adecuada para la aplicación requerida. En la figura 3.4 se puede ver una representación gráfica de como es es el proceso de optimización de cada peso a lo largo de las iteraciones, tratando de ajustar sus valores de tal forma que se lleve el EMC a su mínimo global (idealmente).

Figura 3.4: Representación gráfica de la optimización de pesos en una RNA. [IBMCloudEducation, 2020]

Todo este proceso de afinación de pesos y umbrales es denominado entrenamiento del modelo y el conjunto de datos con etiquetas que indican la respuesta que tiene que dar el modelo para cada uno de estos datos es comúnmente deno-

minado como *dataset*. A su vez el *dataset* típicamente es dividido en 2 subconjuntos, un conjunto *train* para usar durante el entrenamiento y otro conjunto *test* que se usa luego para determinar que tan correctas son las respuestas del modelo respecto a su tarea (usualmente *train* se lleva un porcentaje notoriamente mayor de elementos que *test*).

El entrenamiento de un modelo de RNA está influenciado por dos parámetros, el tamaño de lote (*batch*) y la cantidad de épocas de entrenamiento (*epochs*). *Batch* hace referencia a la cantidad de ejemplos del conjunto *train* que son procesados antes de realizar cualquier modificación en los pesos y umbrales, los resultados de los elementos de un mismo batch son promediados y con esto es que se realiza la optimización (particularmente útil cuando se tienen elementos anómalos en el conjunto de que *train* podrían desequilibrar el modelo). *Epoch* hace referencia a cuántas veces se recorre el conjunto de *train* en su totalidad. La optimización de cada iteración es utilizada para inicializar cada entrenamiento subsiguiente con el conjunto *train*. Por ejemplo, si se entrena una RNA con un *train* de 3648 elementos, con $batch = 64$ y $epoch = 300$ entonces se recorrerá 300 veces el conjunto de datos *train*, modificando 57 veces los parámetros de la RNA en cada iteración.

Una variante particularmente útil de las RNA son las RNC (Redes Neuronales Convolucionales), estas se usan principalmente para detección de objetos, clasificación de imágenes u otros problemas que involucren imágenes en sus *datasets*. Al igual que en las RNA cada nodo de una RNC recibe inputs, los procesa y los reenvía a los nodos de la siguiente capa. Sin embargo, las RNC se caracterizan en que las transformaciones que realizan sus nodos consisten principalmente operaciones de convolución que se aplican sobre las matrices de valores que reciben como valores de entrada, matrices que se corresponden a las representaciones lógicas de los conjuntos de píxeles de las imágenes a procesar. El procesamiento de las matrices iniciales mediante convoluciones en conjunto con el uso de ciertas funciones de activación como ReLU permite la extracción de lo que se conoce como mapas de características, los cuales son los que usa el modelo para lograr su objetivo. El que se hace luego varía un poco según que se quiera hacer, pero por ejemplo, si lo que se buscase es diferenciar imágenes de gatos y perros entonces es probable que se usen alguna función como SoftMax para procesar los mapas de características y con ello obtener la probabilidad de que la imagen original corresponda con un perro o un gato respectivamente. Los ajustes que las RNC van realizando a lo largo de su entrenamiento se ven principalmente en los de los filtros convolucionales usados por cada nodo.

3.3.1. Familia de arquitecturas YOLO

YOLO [Redmon et al., 2015] es una familia de arquitecturas y modelos que tienen principalmente el objetivo de resolver problemas de detección de objetos. Las RNC producidas con este reciben una imagen, buscan en ella objetos previamente especificados y responde un conjunto de BBs (*Bounding Boxes* o Cajas Delimitadoras) según la cantidad de objetos que haya detectado. Las BB son representaciones abstractas que encapsulan información útil para identificar

los objetos de una imagen. En general a cada una le corresponden 5 valores, primero un valor entero que indica la etiqueta correspondiente al objeto que corresponde a la BB (al primer objeto a reconocer se le asigna cero y se avanza de a uno hasta que a todos los objetos les corresponda un valor), segundo y tercero la latitud y longitud del centro del objeto sobre la imagen dada, cuarto y quinto el alto y ancho de la BB respectivamente. Esta información se puede usar por ejemplo para delimitar los objetos de interés en una imagen o vídeo específico, tal como se puede ver en la imagen 3.5. Claro que previo a su uso hay que entrenarla con un *dataset* que contenga un conjunto de imágenes que tengan ejemplos de los objetos que se quiera detectar, cada uno con un archivo de etiquetas que especifique la información de las BBs que le corresponden.

Figura 3.5: a) Imagen de un perro y un pájaro junto con un archivo de etiquetas que especifica los parámetros de las BB que delimita la porción de la imagen que corresponde a cada uno. b) Especificación visual de las BBs del archivo de etiquetas sobre la imagen en cuestión.

La obtención de un *dataset* apropiado para tareas de detección podría llegar a ser complicada, dependiendo de qué es lo que se quiera detectar. Es ideal disponer de imágenes ya con sus archivos de etiquetas correspondientes. Pero, si no se tuviesen los archivos de etiquetado entonces se los puede generar haciendo uso de softwares de etiquetado manual como LabelStudio [Tkachenko et al., 2022] o MakeSense [Skalski, 2019]. También se puede dar que el conjunto de imágenes del que se dispone es bastante limitado (respecto de las cantidades necesarias para entrenar un modelo de detección), una práctica común para solventar este tipo de problemas es aumentar artificialmente la cantidad de datos (práctica mejor conocida como *Data Augmentation*). Las formas de hacerlo varían dependiendo el dominio pero cuando se trata de datos en formato de imágenes lo que se suele hacer es aplicar transformaciones en las mismas, cambios de color, orientación, contraste, entre otros. Más allá de la transformación particular, se busca modificar la imágen original lo suficiente para que parezca otra imagen a ojos de la RNC pero no tanto como para que se pierda la información que interesa de la misma. En la figura 3.6 se pueden ver varios ejemplos de imágenes generadas artificialmente con el fin de incrementar el tamaño de un *dataset*.

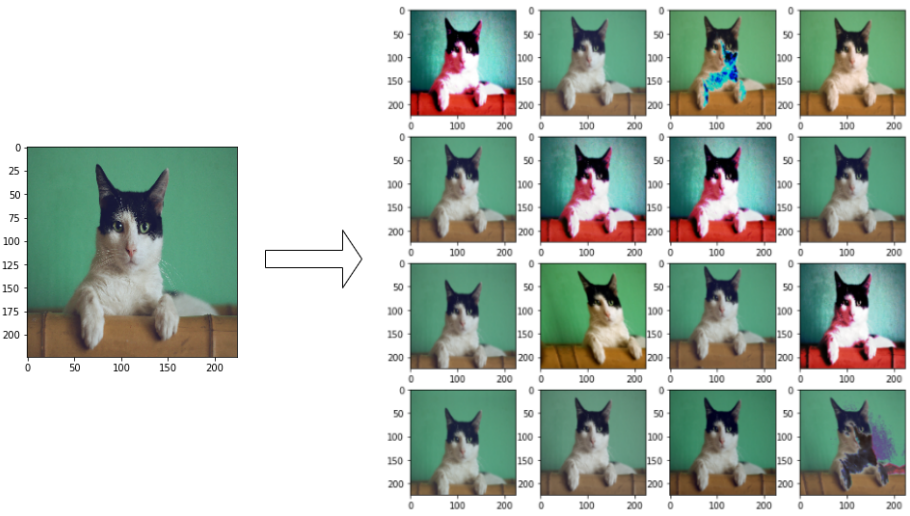

Figura 3.6: Imágenes generadas artificialmente en base a una imagen de un gato [Landup, 2022]

Naturalmente, en los modelos de detección de objetos hay que tener algunas consideraciones especiales, ya que como se indico antes los *datasets* no poseen tan solo las imágenes en cuestión sino que por cada una también se tiene información de etiquetado. Por lo tanto, al incrementar artificialmente la cantidad de imágenes también se debe incrementar de igual manera la cantidad de etiquetas. Si la transformación por la que se generaron las nuevas imágenes fue únicamente de color, contraste o atributos similares que no hayan implicado desplazamien-

tos en los objetos de la imagen, entonces basta con duplicar las etiquetas de la imagen original. Pero, si se realizaron cambios de orientación, inversión de los ejes cartesianos o alguna transformación que haya implicado el desplazamiento de los objetos de la imagen, entonces se deben modificar las etiquetas acorde a los desplazamientos realizados.

Ya con todos estos conceptos vistos es buen momento para empezar a hablar del sistema desarrollado, a partir del capitulo siguiente se empieza se empezara a hablar del mismo y de los detalles de su funcionamiento.

Capítulo 4

Frontend

En el presente capítulo se explica el Frontend del sistema construido y las distintas decisiones que se tomaron en su diseño. Antes que nada se describe de forma breve el sistema objetivo, para tener una idea clara y definida de qué es lo que se buscó durante el desarrollo. Posteriormente se planteará en varias secciones cada cual explica una determinada funcionalidad del Frontend desarrollado.

4.1. Requerimientos

Se busca lograr una aplicación que asista al usuario durante el proceso de digitalización de placas astronómicas. Teniendo esto en cuenta, se analizó el proceso de digitalización como tal y en base a este se definió un conjunto de requerimientos funcionales y no funcionales para el mismo.

En primera instancia, se aislaron las tareas mínimas que el software debía cumplir para asistir al usuario en el proceso de digitalización de placas espectrográficas. En base a estas tareas se definieron los requerimientos funcionales mismo, estos son:

- Carga y selección de datos: El usuario debe poder recuperar y seleccionar los escaneos en formato TIFF de placas espectrografías a procesar.
- Recorte e identificación de espectros: Cada escaneo recibido puede tener de cero a N espectros (siendo N un entero positivo), el sistema debe recortar de forma automática los espectros correspondientes a cada placa para su posterior exportación en archivos individuales. El usuario debe poder editar los recortes realizados en caso de que considere que no son adecuados.
- Carga de metadatos: Como se dijo antes, los recortes de cada espectro se exportan de forma individual pero a cada uno se le debe adjuntar una serie de metadatos propios de las ciencias astronómicas. El software debe ofrecer

la posibilidad de que el usuario cargue esos metadatos, con asistencia y validación.

- Exportación de resultados: Los recortes finales de cada espectro junto a sus correspondientes metadatos se deben exportar en archivos FITS, empaquetando máximo un espectro por archivo. Notar que hay una relación 1:N entre las placas a digitalizar y los archivos que se deben producir, ya que cada placa puede contener varios espectros.

Una vez definido los puntos esenciales del software se decidió acerca de sus requerimientos no funcionales. Inicialmente se partió con ciertos puntos base pero a medida que el software se desarrolló y varios usuarios de la FCAG lo probaron, estos requerimientos fueron cambiando. No obstante, siempre se mantuvo el objetivo final de reducir lo más posible los tiempos de digitalización de placas astronómicas que se manejan actualmente. Al final, los requerimientos no funcionales son los siguientes:

- Automatización de tareas: el sistema debe automatizar los más posible las tareas a realizar, en pos de minimizar las acciones requeridas por el usuario y que la digitalización ocurra lo mas rápido posible.

- Permitir ajustes: las predicciones que el sistema realiza acerca de los recortes de los espectros y los cálculos que el sistema realiza para ayudar en el llenado de metadatos son partes esenciales del sistema y se espera que su funcionamiento sea el adecuado. Aun así, es esperable que en ocasiones haya discordancia entre lo que el sistema considera correcto respecto a lo que el usuario opina. Por tanto, se decidió que el usuario tenga siempre la última palabra, permitiendo que este revise y edite los recortes de espectros indicados por el sistema revisar y todos aquellos metadatos en los que el sistema asista al usuario para su cálculo.

- Visualización de escaneo con sus espectros: para una adecuado trabajo de digitalización es necesario que el usuario pueda visualizar el escaneo que se está digitalizando en todo momento, así como los espectros que se hallan en el mismo.

- Filtros para la visualización de espectros: ocasionalmente algunos detalles de los espectros a analizar no son apreciables a simple vista. Por tanto se requiere que el sistema mantenga opciones que permitan modificar el brillo, contraste y color de la imagen del escaneo que se le muestra al usuario.

- Guardado automático: las interrupciones en medio de los trabajos de digitalización son relativamente comunes, principalmente debido a lo extensos que estos pueden llegar a ser. Por tanto el sistema debe tener un sistema de guardado automático que permita que, ante un cierre del sistema a mitad del procesamiento de un archivo, el usuario pueda volver a abrir la aplicación y continuar el procesamiento donde lo dejó. De esta forma el

usuario puede dejar un procesamiento a medias, irse a su casa y volver al día siguiente para continuarlo donde lo dejó. De forma indirecta, con esto el usuario también debe poder alternar entre los procesamientos de distintos archivos sin perder progreso en ninguno de ellos.

4.1.1. Modelo de solución

Para resolver el funcionamiento de la aplicación se optó por el ya clásico esquema de aplicación web en 2 partes: un Frontend responsable de gestionar la interacción con el usuario y un Backend que encapsula la complejidad de la aplicación y habilita varios métodos de interacción en forma de una API REST, de forma que el Frontend pueda usarlo sin preocuparse por la complejidad interna de su funcionamiento. El Frontend se construyó en base al framework *Node.js* y en las siguientes secciones se explica más acerca de su dinámica y funcionamiento. Por otro lado, el Backend (explicado más a detalle en el capitulo 5) gestiona todo lo relacionado al almacenamiento y persistencia de los escaneos a procesar.

4.2. Carga y selección de datos

En una primera instancia el Frontend debe obtener el listado de los archivos de escaneos disponibles y mostrárselo al usuario para que este seleccione el archivo con el que desea trabajar.

El listado como tal es recuperado con ayuda del Backend a través del *endpoint* "/api/allPaths" que expone en su API. Cada elemento del listado siempre esta en formato TIFF, principalmente debido a como es el proceso de escaneo de cada placa. Para el armado de la visualización es importante que el usuario pueda distinguir con que archivos trabajó anteriormente, por lo tanto, el Frontend divide los elementos del listado en 2 grupos:

- **En Proceso**: agrupa todos aquellos archivos sobre los que hay interés de procesamiento debido a que su digitalización todavía está pendiente.
- **Exportado**: agrupa aquellos archivos que ya fueron procesados en su totalidad y que por ende ya se encuentran exportados. Es importante mostrarlos al usuario, por si este quiere corregir la información que subió en un procesamiento previo.

El backend conoce el estado de procesamiento de cada archivo gracias al sistema de guardado automático que se implementó, donde a medida que un usuario procesa un archivo, el sistema actualiza la información que guarda del mismo.

Entre los archivos TIFF que se tienen que procesar también es de interés que el usuario diferencie cuales todavía no fueron abiertos y cuales no han finalizado todavía su procesamiento, por lo que para evitar navegaciones innecesarias entre múltiples submenús simplemente se optó por una señalización visual, haciendo que los archivos que están a medio procesar resalten gracias a un fondo azul

suave, que hace contraste con el fondo blanco de los demás archivos. Como se puede ver en la figura 4.1, los archivos a medio procesar tienen a su derecha un paréntesis con un valor entero dentro, esto hace referencia a la cantidad de espectros que se identifican en cada uno.

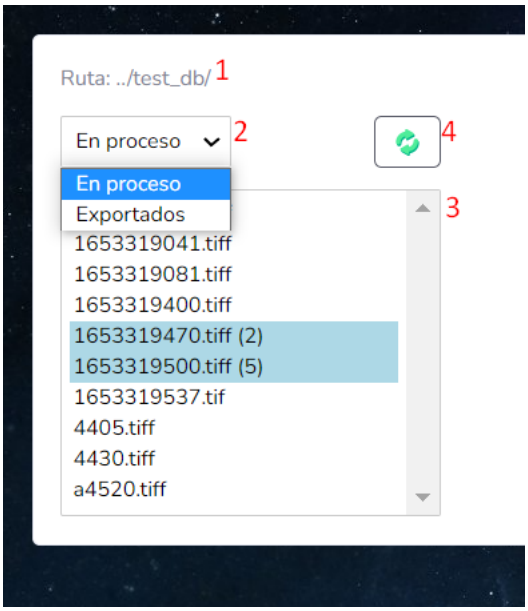

Figura 4.1: Menú de selección del Frontend. 1) Path de la carpeta que el backend está usando para la recopilación de escaneos. 2) Selector desplegable para moverse entre el listado de escaneos que ya fueron exportados y los que no. 3) Listado de los escaneos que corresponden a la opción indicada en el selector, con fondo blanco aquellos espectros para los que no se ejecutó ningún paso del procesamiento, con fondo azul el caso contrario, para estos últimos se especifica entre paréntesis la cantidad de espectros que tienen. 4) Botón de Refrescar para actualizar la lista en caso de que algún espectro ya haya sido procesado o por si se agregaron nuevos.

Una vez el usuario elige con que archivo quiere trabajar el Frontend procede con la recuperación de la información específica del mismo. Entre otras cosas, recupera la imagen del escaneo en cuestión e información técnica del archivo la cual es mostrada debajo del menú de selección de archivo. En los casos en los que el archivo ya haya sido procesado (parcial o totalmente) entonces también se recuperan todos los metadatos que se tengan del mismo y la información correspondiente a las BB de los espectros que se identificaron en su escaneo. La utilidad de este último par de conjuntos de datos será explicada mas adelante, pero es relevante de cara a permitir que el usuario prosiga con el procesamiento de un archivo. La recuperación como tal de esta información se hace mediante el *endpoint* "/api/image/load" que expone el Backend en su API. Según el

contenido de la información obtenida se procede de una u otra forma, esto será explorado en las secciones siguientes, pero a modo de adelanto y para tener una visión general de la aplicación (antes de involucrarse en las explicaciones de cada componente) en la figura 4.2 se muestra que es lo que ve el usuario al momento de seleccionar un archivo.

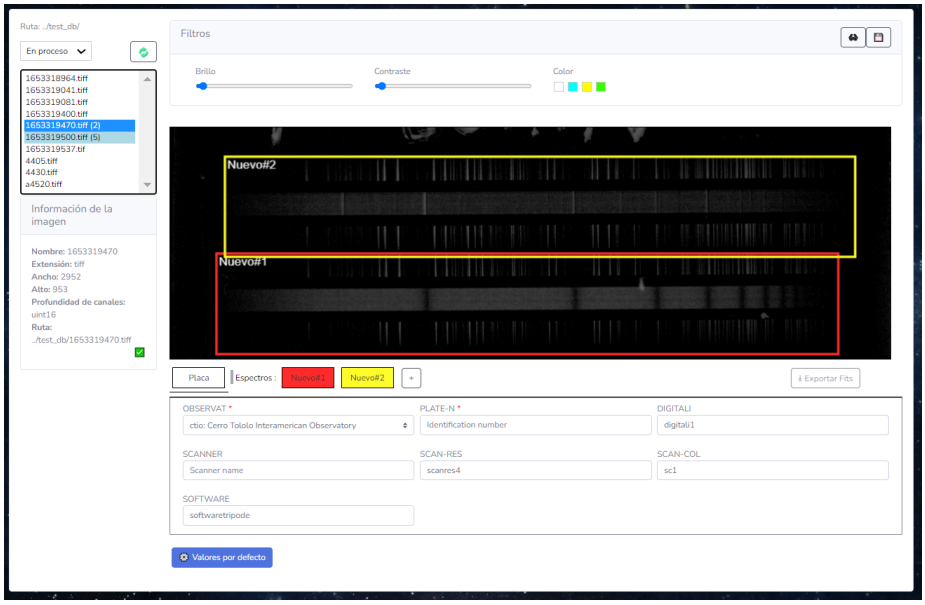

Figura 4.2: Vista general del Frontend al momento de seleccionar un archivo. A la izquierda el listado de archivos junto los datos específicos del archivo que se encuentra seleccionado. A la derecha arriba está la interfaz de edición de recortes y a la derecha abajo está el menú de llenado de metadatos junto con las opciones de exportación.

4.3. Recorte e identificación de espectros

Una vez se tiene la información del archivo con el que se quiere trabajar entonces el Frontend carga el menú de procesamiento de escaneos, el cual contiene, entre otras cosas, la ventana de recorte e identificación de espectros. Para ello el Frontend identifica cuál es la estado de procesamiento del archivo a tratar y dependiendo del mismo toma una decisión u otra:

- El archivo nunca ha sido abierto: al recuperar la información del archivo no se obtuvo ningún dato respecto a procesamientos anteriores, por lo que el Frontend solicita al Backend que use el modelo de detección de espectros para inferir las BBs que corresponden a los espectros del escaneo en

cuestión. Para hacer uso del modelo de inferencia de espectros el Backend expone el *endpoint* "/api/predict" en su API.

- El archivo se encuentra a medio procesar o ya ha sido exportado: al recuperar la información del archivo se recuperó la información de los BBs y metadatos que quedaron de sus procesamientos anteriores, por lo que simplemente se los toma para poder continuar el procesamiento donde queda en la última ocasión.

Lo que ocurre durante el proceso de carga de esta ventana es invisible al usuario. Éste solo requiere la carga de la ventana de recorte e identificación de espectros, por lo que el Frontend en base al conjunto de BBs que obtuvo carga la ventana en cuestión, permitiendo que a través de la misma el usuario visualice los espectros identificados y que los modifique en caso de que así lo requiera. En la figura 4.3 se pude ver una representación gráfica del proceso que siguen el Frontend y el Backend para mostrar al usuario la información de un escaneo específico.

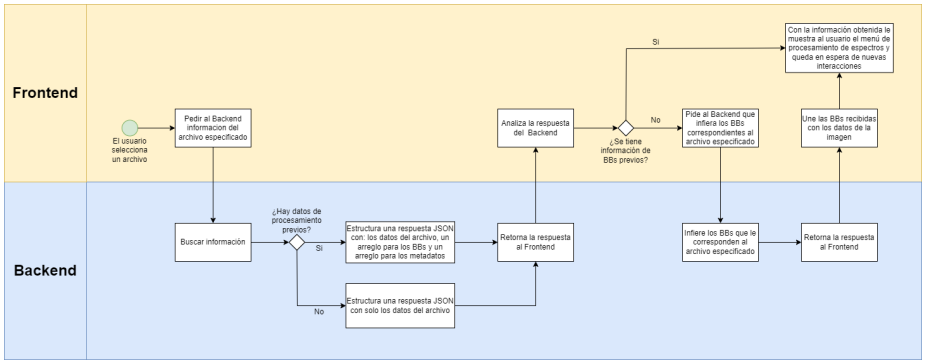

Figura 4.3: Secuencia de acciones que ocurren al momento de solicitar los datos de un espectro.

4.3.1. Sistema de modificación de Bounding Boxes

Naturalmente, el sistema no es infalible y puede darse que el usuario quiera realizar cambios en los BBs que el sistema calcula. En consecuencia, se incorporó al software un sistema de edición de BBs que permite que el usuario modifique tanto la cantidad de BBs como sus dimensiones. Su concepto es sencillo, se muestra la imagen original y sobre esta se despliegan varias cajas que se corresponden a las BBs que se tienen en ese momento, estas cajas son editables por el usuario y los cambios que realiza sobre estas son reflejados en la información de las BBs del escaneo. El sistema de edición implementado permite al usuario:

1. Modificar la posición de un BB: haciendo click y arrastrando la caja correspondiente.

2. Modificar las dimensiones de un BB: haciendo click y arrastrando sobre los vértices y/o aristas de la caja correspondiente.

3. Eliminar BBs: cada caja que corresponde a un BB posee un pequeño símbolo de "x" en su esquina superior derecha para eliminarlo.

4. Añadir BBs: debajo de la imagen del escaneo se muestra el menú de llenado de metadatos, el mismo será introducido mas adelante pero se menciona por que en su parte superior se muestra una solapa por cada BB identificada. El elemento mas a la derecha del listado siempre es un botón con un símbolo de -", si el usuario hace click en el este entonces se agrega un nuevo BB junto con su caja correspondiente y la solapa que le corresponde en el menú de llenado de metadatos. Naturalmente, no se sabe dónde es que el usuario cree que es necesaria la nueva BB, por lo que su caja simplemente es desplegada en el centro de la imagen para que el usuario la mueva según su conveniencia.

También se incorporó un sistema de filtros que permite cambiar el brillo, contraste y color base de la imagen del escaneo que muestra el Frontend. Fue añadido para preservar la comodidad del que esta inspeccionando la imagen, ya que dependiendo del escaneo puede llegar a ser mas o menos difícil diferenciar a simple vista los espectros de la misma. También se incorporó un botón para alternar el uso de los filtros y otro que guarda la configuración actual de los filtros por si el usuario desea mantenerla al moverse a otros escaneos. En la figura 4.4 se muestra el menú de recorte e identificación de espectros junto con la interfaces correspondientes a las funcionalidades antes descritas.

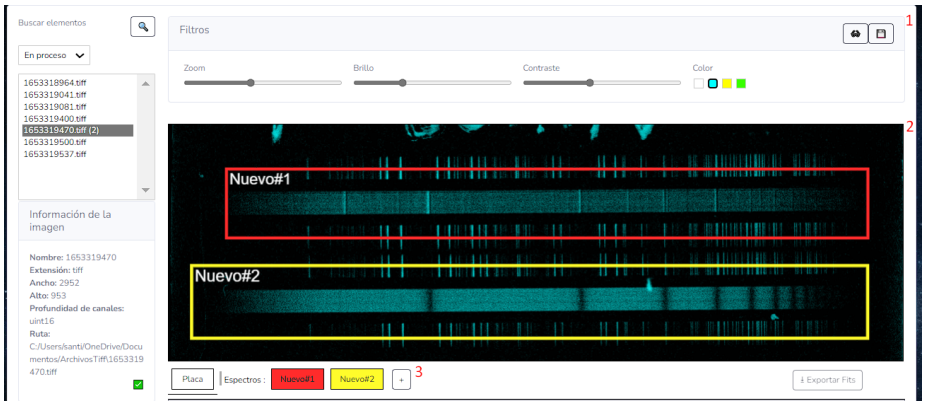

Figura 4.4: Menú de recorte e identificación de espectros. 1) Menú de selección de filtros. 2) Imagen del escaneo a digitalizar junto con sus correspondientes BBs. 3) Botón para añadir una nueva BB.

4.4. Llenado de metadatos

Para procesar un escaneo y obtener varios archivos FITS, uno por cada espectro que contenga, se necesitan dos elementos principales de cada espectro:

1. Una imagen del espectro en cuestión, este elemento se obtiene en base a las BBs que señalan cada espectro (su proceso de obtención fue explicado en la sección 4.3).

2. Un conjunto de metadatos que especifique toda la información que interesa persistir del espectro.

Sobre este último elemento trata esta sección, sobre qué metadatos se requieren y cómo es que el usuario y el sistema pueden colaborar para su obtención.

4.4.1. Definición de metadatos a recopilar

No hay ningún estándar que especifique qué campos hay que adjuntar a una placa espectrográfica cuando esta es convertida al formato FITS, pero si hay algunas guías de formato mas generales que definen algunas normas [Wells et al., 1981]. En base a este tipos de guías y recibiendo asesoramiento externo de profesionales del campo se definió un conjunto de metadatos que cumple con los requisitos mínimos de los archivos FITS y posee un conjunto adecuado de información astronómica para el uso de otros investigadores.

Para definir el conjunto primero se recopiló el listado de metadatos mínimos que se espera de un FITS, los cuales son:

- SIMPLE: Variable lógica que indica si el archivo cumple con el estándar básico de FITS. En general "T" significa que lo hace y "F" que no. En este caso es de interés que cumpla, así que siempre se asigna "SIMPLE=T".

- BITPIX: Variable entera que especifica el número de bits usados para representar el valor de los píxeles. Como es evidente a mas bits se usen mayor sera su profundidad de color. Para procesar los escaneos se decidió usar "BITPIX=16".

- NAXIS: Variable entera que especifica el número de ejes coordenados en la imagen. Coincide con la cantidad de dimensiones de la información que se quiere almacenar. En este caso son imágenes bidimensionales por lo que siempre se tendrá "NAXIS=2".

- NAXIS1: Variable entera que especifica el número de píxeles a lo largo del eje que varía más rápidamente en el arreglo (Ancho en píxeles de la imagen).

- NAXIS2: Variable entera que especifica el número de píxeles a lo largo del segundo eje que varía rápidamente en el arreglo (Alto en píxeles de la imagen).

Esos son los datos mínimos del formato, pero con el paso del tiempo se han ido agregando otros campos opcionales (de uso recomendado) que añaden información útil a los archivos. De los campos que se suelen añadir se agregaron al conjunto los que encajan con la situación de las placas astronómicas, siendo estos:

- OBJECT: Identificador de objeto astronómico.
- OBSERVAT: Observatorio donde la placa fue concebida.
- DATE-OBS: Fecha de observación.
- UT (Universal Time): Tiempo universal (hh:mm:ss) en el que la placa fue concebida.
- RA (Right Ascension): Coordenada útil ubicar para identificar una posición del cielo.
- DEC (Declination): Coordenada útil ubicar para identificar una posición del cielo.
- RA2000: Campo RA pero tomando como referencia el año 2000.
- DEC2000: Campo DEC pero tomando como referencia el año 2000.
- EXPTIME: Tiempo en segundos que el espectro estuvo expuesto.
- DETECTOR: Tipo de emulsión o cámara utilizada para la captura del objeto astronómico.
- IMAGETYP: Tipo de imagen.
- OBSERVER: Nombre del observador.
- EQUINOX: Referencia astronómica tomada al momento de tomar la placa. De interés para describir RA y DEC.

También, luego de consultar con expertos en el tema se decidió incorporar varios campos fuera del estándar, los cuales se considera que añaden información astronómica útil. Estos son:

- TIME-OBS: Tiempo local (hh:mm:ss) al inicio de la observación.
- ST (Sidereal Time): Tiempo sidereal medio local.
- HA (Hour Angle): Ángulo horario.
- RA1950: Campo RA pero tomando como referencia el año 1950
- DEC1950: Campo DEC pero tomando como referencia el año 1950.
- GAIN: Ganancia.

- DIGITALI: Nombre del Digitalizador.
- SCANNER: Nombre del escáner utilizado.
- SOFTWARE: Nombre del software de escaneo utilizado.
- PLATE-N: Numero de identificación de la placa donde estaba el espectro en cuestión.
- MAIN-ID: Identificador del objeto.
- SPTYPE: Tipo espectral.
- AIRMASS: Masa de aire media del objeto.
- JD (Geocentric Julian Day): Dato para la ubicación temporal.

Notar que algunos de los campos antes descritos hacen referencia a datos de la placa astronómica y otros a datos de los espectros que esta contiene. Se sabe que una misma placa puede contener múltiples espectros pero los archivos FITS que se quiere generar son individuales según el espectro. Por ello, es bastante probable que al procesar una placa los datos que corresponden a la misma se tendran que repetir en cada uno de los archivos de los espectros que contiene. Por lo tanto, en pos de minimizar la cantidad de veces que el usuario tiene que dar la misma información, se realizó una clasificación de los metadatos según su correspondencia. En esta se tuvieron en cuenta 3 grupos de metadatos:

1. Metadatos del archivo: aquí se agrupan aquellos metadatos que contienen información relacionada a las características técnicas del archivo. Son deducibles en base a la inspección del archivo TIFF a digitalizar.

2. Metadatos de la placa: en este grupo están incluidos los metadatos que repiten su valor en todos los espectros de una misma placa, sus valores son independientes de los espectros que la placa contiene.

3. Metadatos del espectro: los metadatos de este grupo pueden variar su valor entre distintos espectros, por lo tanto se tienen que completar por separado en cada uno de los espectros a procesar.

En el cuadro 4.1 se muestra a qué categoría corresponde cada elemento del conjunto.

Metadatos:		
De cada Archivo	De la Placa	Del espectro
SIMPLE	OBSERVAT	OBJECT
BITPIX	OBSERVER	DATE-OBS
NAXIS	DIGITALI	TIME-OBS
NAXIS1	SCANNER	UT
NAXIS2	SOFTWARE	ST
	PLATE-N	HA1
		RA
		DEC
		GAIN
		RA2000
		DEC2000
		RA1950
		DEC1950
		EXPTIME
		DETECTOR
		IMAGETYP
		SPTYPE
		MAIN-ID
		AIRMASS
		JD
		EQUINOX

Cuadro 4.1: Clasificación de los metadatos segun la entidad a la que corresponden.

4.4.2. Diseño del formulario de carga de metadatos

El recuperar la información de cada metadato y calcularlos cuando es necesario, es probablemente el paso que mas tiempo requiere en un proceso de digitalización típico, son muchos los metadatos a asignar, cada uno con sus propios valores y métodos de cálculo. Por tanto, su optimización es vital, en ella radica la rapidez en la que el usuario con ayuda del sistema puede llegar a procesar cada archivo. A continuación se explora cómo es que se diseñó el formulario de carga de metadatos y como este asiste el usuario durante el procesamiento, en pos de acelerar el proceso lo máximo posible.

Como se dijo en la sección anterior, los metadatos del conjunto *Archivo* (cuadro 4.1) son deducibles en base al análisis del archivo TIFF que se debe procesar. Ya que la mitad de los datos corresponden a valores fijos y la otra mitad hace referencia a las dimensiones en píxeles del escaneo a procesar se resolvió que el Frontend los resuelva tras bastidores, sin que el usuario tenga que intervenir en su llenado. Los datos fijos meramente son asignados con el valor que les corresponda y los datos relativos a las dimensiones de la imagen son facilitados por el Backend al recuperar la información de la imagen. El Backend calcula estos datos simplemente haciendo un análisis vía script del

archivo en cuestión. Naturalmente, los datos relativos a las dimensiones de la imagen podrían llegar a ser de interés para el usuario por lo que son mostrados junto al resto de información básica del archivo, debajo del listado de archivos disponibles, tal como se puede ver en la figura 4.2.

Respecto al conjunto de metadatos correspondiente a la etiqueta *Placa*, sus metadatos requieren información no deducible de forma automática. Por lo tanto, para procesarlos se resolvió que el usuario llene un formulario donde especifica de forma manual sus valores. Para el ingreso de cada valor se dispuso campo de texto editable, a excepción del campo OBSERVAT, en el cual se utilizó un selector debido a que para elegir su valor se tiene que respetar un listado de observatorios dado por el Backend. El uso del formulario *Placa* permite que el Frontend repita sus valores para cada exportación FITS a realizar. Si no estuviera, el usuario tendría que repetir sus valores en los formularios correspondientes a cada espectro.

Por último, el conjunto de metadatos correspondiente a la etiqueta *Espectro.* se debe completar por cada uno de los espectros a procesar, ya que no hay nada que asegure que sus valores sean los mismos de un espectro a otro. La situación para completar sus metadatos es una mezcla de la de los dos conjuntos anteriores. Algunos de los metadatos del conjunto son deducibles pero sólo si se tiene cierto conjunto de metadatos para cada uno. Juntando todas las relaciones de dependencia se puede armar una jerarquía que permite identificar los metadatos necesarios para calcular cada uno de estos. En la raíz de la misma se tienen los metadatos OBJECT, DATE-OBS, OBSERVAT y UT, que no son calculables en base a otros metadatos y ademas son necesarios para calcular los demás. Notar que la situación de OBSERVAT es un poco especial, ya que este pertenece al conjunto *Placa*. En la figura 4.5 se puede apreciar la jerarquía mencionada. Los metadatos EXPTIME, IMAGETYP, OBSERVER y DETECTOR también pertenecen al conjunto *Espectro* pero se los mantiene fuera de la jerarquía ya que no son calculables y tampoco son necesarios para calcular otros metadatos. Teniendo todo lo antes descrito se decidió que para resolver el llenado de metadatos del conjunto *Espectro* se haría uso de un formulario por cada BBs dispuesta en el menú de recorte e identificación de espectros. Cada formulario forma una relación 1:1 con sus BBs correspondientes, cuando una se crea también se crea su respectivo formulario, cuando una se borra también su formulario. Inicialmente, se requiere que el usuario provea los cuatro metadatos raíz para determinar todos los metadatos derivados. Para ello, cada formulario muestra los campos correspondientes a los metadatos OBJECT, DATE-OBS y UT para que sean completados por el usuario. El valor de OBSERVAT es extraíble del formulario *Placa* y por tanto no se requiere que el usuario lo ingrese otra vez. Una vez los valores están dispuestos, el Frontend habilita en el formulario un botón con la leyenda "Buscar Metadatos". Con el, se desencadena una solicitud al Backend para que calcule en base a los metadatos raíz todos los metadatos derivados que pueda y se los responda al Frontend.

Para esta tarea el Backend expone el *endpoint* "/api/getMetadata". Una vez se tiene el listado de metadatos calculados, el Frontend expande el formulario con el que se realizó la búsqueda, para que también contenga campos para los

demás metadatos del conjunto *Espectro*, los campos de los que se obtuvo un resultado son rellenados automáticamente con el valor obtenido, aunque manteniéndolos modificables por si el usuario quiere acomodar su valor. En ocasiones el usuario no dispone de todos los metadatos raíz, para esos casos se habilitó que se les pueda asignar el valor "MISSING", de esta forma el Backend puede reconocer su ausencia y evita calcular los metadatos que dependen de los metadatos raíz faltantes. Aquellos campos que no pudieron ser calculados simplemente se los deja vacíos para que sean determinados por el usuario. Dadas sus características EXPTIME, IMAGETYP, OBSERVER y DETECTOR siempre tendrán que ser determinados por el usuario. Para que el usuario pueda distinguir de forma rápida la situación de cada metadato se colorean de azul los campos auto completados y de blanco los que tienen que ser completados por el usuario.

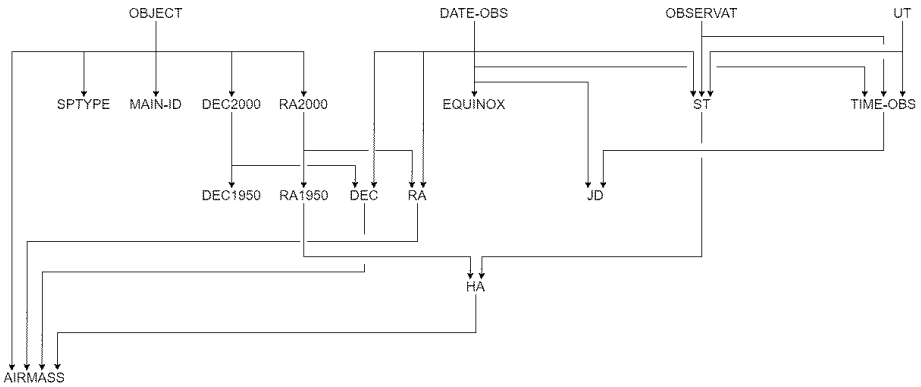

Figura 4.5: Visualización de las dependencias de los distintos metadatos. Cada metadato puede ser determinado si se poseen los metadatos de los niveles superiores del gráfico con los que está conectado. Los metadatos que no aparecen en el gráfico no dependen de ningún metadato pero tampoco son necesarios para determinar otros.

Por cada escaneo a procesar se tiene que completar un formulario de placa y por cada BB en el escaneo un formulario de espectro. Desplegar todos los formularios uno debajo de otro no es una buena opción, ya que el usuario no podría analizar la imagen del escaneo a procesar al mismo tiempo que completa los formularios que quedaron más abajo. Para solucionar este problema y optimizar la navegación lo más posible, se agruparon los distintos formularios bajo un sistema de solapas ubicado justo debajo debajo del menú de recorte e identificación de espectros. Su funcionamiento es similar a como funcionan las pestañas de un navegador, se despliega un listado de solapas en la parte superior del menú de llenado de metadatos, cada solapa es relacionada a un formulario concreto y el usuario puede navegar entre los distintos formularios seleccionando la solapa correspondiente a cada uno. En general, la primera solapa siempre corresponde al formulario de la placa y las subsiguientes se corresponden con los

formularios de espectro. De esta forma el llenado de metadatos puede ser realizado sin perder de vista la imagen del escaneo que se está procesando, además de que se puede navegar entre los distintos formularios sin tener qué pasar por todos los formularios que haya entre los dos. Para no confundir que formulario le corresponde a cada espectro, se usa el campo OBJECT como identificador, teniéndolo tanto cada BB como la solapa que le corresponda. Además, se los colora a ambos del mismo color.

También, en pos de mejorar la usabilidad de la aplicación se desarrollo una funcionalidad especial para los campos de los metadatos OBSERVER, IMAGETYPE, DIGITALI, SCANNER, SCAN-RES, SCAN-COL y SOFTWARE. Al procesar un conjunto de placas sus valores se suelen mantener invariantes en cada uno de los archivos u oscila entre 2 o 3 valores comunes. Por lo tanto se permite especificar un listado de valores comunes para cada unos de estos campos en conjunto con la selección de uno de estos como valor por defecto. De esta forma estos campos se llenarán automáticamente al iniciar el procesamiento de un nuevo archivo, según lo que especifique su valor por defecto. En caso de que el valor por defecto no corresponda en un valor específico, el usuario puede modificarlo de forma manual o través de un selector. Tanto el sistema de navegación entre formularios como el botón "Valores por defecto" son apreciables en la figura 4.6.

Figura 4.6: Visualización del menú de llenado de metadatos. 1) Listado de solapas que permiten acceder a los distintos formularios. 2) Espacio de visualización del formulario seleccionado. 3) Botón para la configuración de valores por defecto.

4.5. Exportación de resultados

Cuando el estado actual de un formulario es apto para exportar se muestra un tilde verde en su respectiva solapa. Una vez todos los formularios se han completado, se habilita el botón "Exportar FITS" (figura 4.5). Al presionarlo,

el Frontend recopila la información necesaria (los BBs junto a los metadatos correspondientes a cada uno) y la información de identificación del archivo en cuestión y envía todo al Backend para que este genere y almacene los archivos FITS que correspondan. Para ello el Backend expone en su API el *endpoint* "/api/generatefits".

Figura 4.7: a) Captura del procesamiento de un archivo sin ningún formulario listo y el botón "Exportar FITS" deshabilitado. b) Captura del procesamiento de un archivo con todos los formularios listos y el botón "Exportar FITS" habilitado.

4.6. Sistema de guardado automático

El uso del sistema supuso una mejora sustancial en el tiempo que se tarda en digitalizar cada archivo. En nuestras pruebas iniciales, el período de procesamiento fue reducido de una hora por archivo a aproximadamente 15 minutos. Aun así, para hacer más amigable la aplicación se incorporó un sistema de guardado automático. Con el, los usuarios pueden cerrar el programa a mitad de una digitalización con la confianza de que mas tarde podrá retomarla donde la dejaron. El autoguardado, se presento particularmente útil para aquellos casos en los que un usuario no tiene cierta información que necesita para terminar de procesar un escaneo determinado, con este pueden dejar pausados aquellos espectros de los que le falta información para retomarlos cuando tenga todo lo necesario, en el medio puede continuar con el procesamiento de otros espectros si así lo desea. Además, el guardado automático impide que los progresos realizados se pierdan ante un repentino corte de luz.

El funcionamiento del sistema de autoguardado es el siguiente: cada vez que un usuario realiza un cambio (en los BB o en los formularios) el Frontend envía todos los datos del archivo actual (metadatos y BB) al Backend (gracias al *endpoint* "/api/image/save"), el cual salva el estado actual del procesamiento del escaneo del lado del Servidor. Así, en caso de se interrumpa el procesamiento de un escaneo (por la razón que sea), los datos del mismo no se perderán y

el usuario podrá retomar el procesamiento cuando convenga. Para retomar el procesamiento de un archivo específico basta abrir la aplicación y seleccionar en el listado el archivo del que se quiere continuar su procesamiento. Una vez se selecciona el archivo, el Frontend la cargara su información desde el Backend.

Naturalmente, el salvado del estado del procesamiento de un archivo no es instantáneo por lo que se incorporó una señal visual junto a los datos de la imagen para que el usuario pueda distinguir si los datos que modificó ya están respaldados o no. La misma es una rueda de carga que luego de recibir una respuesta afirmativa del Backend se cambia a un tilde afirmativo, indicando que es seguro cerrar el programa en ese instante. Dicha señal es apreciable en la figura 4.8.

Figura 4.8: a) Información del archivo actual mientras se realiza el proceso de guardado automático. b) Información del archivo actual luego de que finaliza el proceso de guardado automático.

Hasta aquí llega la descripción del Frontend, en el capítulo siguiente se describe en detalle el Backend.

Capítulo 5

Backend

El Backend centraliza toda la lógica necesaria para el Manejo, Administración y Procesamiento automático de Espectros. El mismo fue escrito en Python, utiliza el framework Flask para la codificación de los *endpoints* e incorpora varias librerías necesarias para su funcionamiento general. En específico, utiliza las librerías:

- AstroPy: Facilita varias funciones útiles para el cálculo de coordenadas astronómicas y conversión de unidades.
- Astroquery: Para realizar consultas a la base de datos de información astronómica SIMBAD [Wenger, M. et al., 2000].
- Astroplan: Facilita funciones de posicionamiento astronómico y operaciones derivadas.
- TimezoneFinder: Para determinar zonas horarias específicas según las coordenadas disponibles.
- PyTorch: Necesaria para realizar inferencias con modelos de detección de objetos YOLO.
- OpenCV: Útil para la manipulación y redimensionado de imágenes.

El presente capítulo se centra en explicar las características principales del Backend construido. En la sección 5.1 se muestra un listado de los *endpoints* que componen la API ofrecida por el Backend. En la sección 5.2 se explica cuál fue el método utilizado para resolver el almacenamiento de información de la aplicación. En la sección 5.3 se da una explicación mas detallada de como el Backend resuelve el cálculo de metadatos ofrecido por el *endpoint* "api/getMetadata". Por último, en la sección 5.4 se como se resuelve el reconocimiento de espectros en imágenes, requerido por el *endpoint* "api/predict".

5.1. API

El Backend expone en su API un conjunto de 7 *endpoints*, útiles para el funcionamiento general del Frontend. A continuación se muestra una descripción breve de cada uno:

- (GET) "/api/allPaths": Recibe una ruta de almacenamiento y retorna un listado de todos los archivos TIFF que haya en la misma.
- (GET) "/api/loadConfig": Informa varios datos útiles para cargar correctamente la aplicación. Actualmente, informa el listado de observatorios a mostrar en el campo OBSERVAT y el Path de donde el backend está buscando los archivos TIFF.
- (GET) "/api/image/load": Recibe el nombre de un escaneo y un determinado path. En caso de que exista, retorna la información del escaneo solicitado.
- (PUT) "/api/image/save": Recibe el nombre de un escaneo, un listado de BBs y un listado de metadatos. El Backend guarda toda esta información para que pueda ser consultada en un futuro. Si ya se tenía información guardada respecto al escaneo especificado entonces se la sobrescribe. Este *endpoint* es usado por el Frontend para hacer funcionar el sistema de guardado automático.
- (POST) "/api/getMetadata": Recibe un listado de metadatos. En base al mismo, calcula todos los metadatos derivados que puede y retorna el listado resultante (Mas detalles respecto a su funcionamiento en la sección 5.3).
- (POST) "/api/predict": Recibe el nombre y ruta de un escaneo. Si existe, genera predicciones del mismo respecto a los espectros que posee. Devuelve la información correspondiente a las BBs halladas, sus dimensiones y sus posiciones (Mas detalles respecto a su funcionamiento en la sección 5.4).
- "/api/generatefits": Genera los archivos FITS correspondientes a la información recibida y los almacena.

En caso de que se desee ver una descripción mas detallada de cada *endpoint* esta puede ser encontrada en el apéndice A.

5.2. Organización de los archivos

Para una comprensión más profunda del sistema es útil conocer cómo este resuelve el almacenamiento de los datos que debe persistir. Hay tres tipos de datos que maneja el sistema: los archivos TIFF iniciales, los datos de persistencia del procesamiento de cada archivo y los archivos FITS resultantes de finalizar el procesamiento de cada archivo.

En el sistema, del lado del servidor se genera una carpeta con todos los archivos TIFF a procesar. La ruta a la misma es especificada en el archivo de configuración "db.json", específicamente en el atributo "workspace_path". Todos los datos de configuración son consultables por el Frontend mediante el *endpoint* "/api/loadConfig" y este puede ver el listado de los archivos TIFF y recuperar la información de cada uno mediante los *endpoints* "/api/allPaths" y "/api/image/load" respectivamente. La distinción de los distintos archivos TIFF dentro de la carpeta no es un problema, ya que desde su creación vienen nombrados con identificadores únicos. Aparte de los archivos TIFF, también se generan dentro de la ruta varias subcarpetas cada una dedicada a cierto tipo de información que al sistema le interesa persistir:

- **output**: Carpeta dedicada a almacenar los archivos FITS generados. En pos de evitar colisiones de nombre, cada FITS generado se nombra siguiendo el patrón "a_b" siendo "a" el nombre del archivo TIFF del que proviene el escaneo y "b" el valor del campo OBJECT del espectro que correspondiente al FITS en cuestión (se sabe que este par de valores no sufre repeticiones).
  Cada FITS trae consigo un PNG del espectro que le corresponde y un TXT que tiene una descripción formal del FITS en cuestión. Ambos archivos tienen el mismo nombre que el FITS pero con su correspondiente extensión. El único *endpoint* que interactúa con esta carpeta es "/api/generatefits".

- **.train**: En esta carpeta se almacenan las imágenes de cada TIFF que fue enviado al Frontend, en formato PNG. Es utilizada principalmente como una caché, que mejora el rendimiento de "/api/image/load" y "/api/predict". Esta carpeta es por defecto oculta ya que no cumple funciones para el usuario final.

- **cache**: Aquí es donde se almacenan los datos de procesamiento (Metadatos y BBs) de los archivos TIFF de los que se haya iniciado su procesamiento. Según que tan procesado está el archivo sus datos se guardan en distintas subcarpetas:

  - saved: Datos de archivos TIFF ya procesados, de los cuales se realizó su correspondiente exportación. Interesa guardarlos por si el usuario considera que es necesario modificar y volver a exportar sus datos.
  - working: Datos de archivos TIFF que están a medio procesar, archivos en los que se está trabajando pero todavía no fueron exportados.

  Los datos de procesamiento de cada archivo TIFF (metadatos y BBs) son persistidos en formato JSON. Este formato es sencillo y de fácil uso para el Frontend. Esta carpeta es de particular interés para los *endpoints* "/api/image/load", "/api/image/save", "/api/getMetadata" y "/api/generatefits".

5.3. Cálculo de metadatos

La situación presentada por el *endpoint* "api/getMetadata" fue moderadamente mas compleja de resolver que la de los demás *endpoints* (a excepción de "api/predict") por lo que se considera adecuado explicar las particularidades de su funcionamiento.

Primero, hay que repasar el funcionamiento que se espera del mismo, este tiene el propósito de facilitar al Frontend el cálculo de metadatos derivados, permitiendo que este se abstraiga de la complejidad que esta tarea implica. A través del *endpoint*, el Backend tiene que recibir el conjunto de metadatos raíz de los que el Frontend dispone, calcular con ellos todos los metadatos derivados que pueda (figura 4.5) y entregar como respuesta el conjunto resultante.

Para la recepción de metadatos iniciales, se resolvió que el *endpoint* espere como entrada los cuatro metadatos raíz con los que se pueden calcular todos los demás metadatos de la jerarquía (OBJECT, OBSERVAT, DATE-OBS y UT), siendo que estos también son los metadatos mínimos que el Frontend requiere del usuario para realizar el cálculo de metadatos derivados. Para aquellos casos en los que no se tenga alguno de estos valores se contempló que estos tomaran el valor "MISSING" para señalarlo.

A partir de los metadatos raíz, existe un total de 14 metadatos derivados cada uno con su propia forma de calcularse. No se consideró adecuado hacer los cálculos para todos en una misma función, por lo que se construyó una clase por cada uno de los metadatos a calcular, cada una manteniendo una función "update()" que recibe el listado de metadatos que se tiene y agrega en caso de poder el metadato que le corresponde. En pos evitar cualquier tipo de incoherencia se hizo que todas estas clases implementen una interfaz común con la firma de la función "update()".

En general, todos los metadatos son determinables en base a la aplicación de operaciones aritméticas sobre otros de los metadatos ya calculados. No obstante algunos metadatos como MAIN_ID, RA2000, DEC2000 o SP_TYPE suelen estar definidos en bases de datos astronómicas como SIMBAD [Wenger, M. et al., 2000]. Se hizo uso de la librerías como Astroquery para consultarlos en la DB ya mencionada. También, en aquellos metadatos que lo requerían se hizo uso de las librerías AstroPy, AstroPlan y TimeZoneFinder para poder poder ejecutar de forma sencilla cálculos y operaciones propios del ámbito astronómico.

5.4. Detección automática de espectros

Como se mencionó antes, el *endpoint* "api/predict" recibe el nombre y ruta de un escaneo e intenta predecir las cantidad, dimensiones y posiciones de los espectros que el escaneo en cuestión posee, respondiendo las predicciones realizadas al Frontend para que este se las pueda mostrar al usuario. No obstante, todavía no se especifico como es que estas predicciones son realizadas, por lo que la presente sección está dedicada a esclarecer este aspecto.

5.4.1. Modelo de Solución

Para enfrentar el problema del reconocimiento de espectros en las imágenes de un escaneo se optó por un enfoque orientado a la IA. Concretamente con la familia de arquitecturas y modelos de detección de objetos YOLO [Redmon et al., 2015].

La premisa de este tipo de modelos se basa en la posibilidad de entrenar un modelo con un conjunto de imágenes previamente seleccionadas las cuales tienen etiquetas con que diferencian los objetos que se quiere distinguir. Una vez entrenado un modelo este puede ser empleado para reconocer los objetos requeridos en imágenes nuevas que no pertenezcan al *dataset* original. En particular, se entrenó un modelo para el reconocimiento de espectros en imágenes de placas espectrográficas, por lo que naturalmente se tuvo que seguir los pasos correspondientes para generarlo.

5.4.2. Preparación del conjunto de datos

Para generar el *dataset* a utilizar se recuperaron 256 escaneos de placas espectrografías que habían sido previamente producidos por la FCAG. Luego, con asistencia de astrónomos con conocimiento del campo y el software de etiquetado LabelStudio [Tkachenko et al., 2022], se realizó el proceso de etiquetado de los distintos espectros de cada escaneo. Se prestó especial atención a no marcar como válidos aquellos espectros que tuviesen defectos significativos (posibles defectos de las placas explicados en la sección 2.2.4).

Una vez que todas las imágenes fueron etiquetadas se procedió con la exportación de las etiquetas generadas. El formato de etiquetas que YOLO requiere es relativamente sencillo, por cada imagen etiquetada se tiene que tiene que tener un archivo TXT que guarda la información correspondiente a cada BB de la misma. De cada BB se guarda: el numero correspondiente a la etiqueta de clase en cuestión (en este caso siempre es 0 ya que solo hay una etiqueta), la latitud correspondientes al centro de la BB, su longitud, el alto de la BB y su ancho. Afortunadamente la mayoría de softwares de etiquetado soportan la exportación de este tipo de archivos de etiquetado, por lo que no se tuvo mayores problemas en ese aspecto. En la figura 5.1 se puede ver una imagen etiquetada junto con el archivo de etiquetas que le corresponde.

El conjunto de los archivos con información de etiquetado fue separado en una carpeta denominada "labels" mientras que el conjunto de imágenes se mantuvo en una carpeta llamada "images". Al final, los escaneos del *dataset* contenían un promedio de 3 espectros cada uno, por lo que se obtuvo un total aproximado de 800 espectros etiquetados de forma manual..

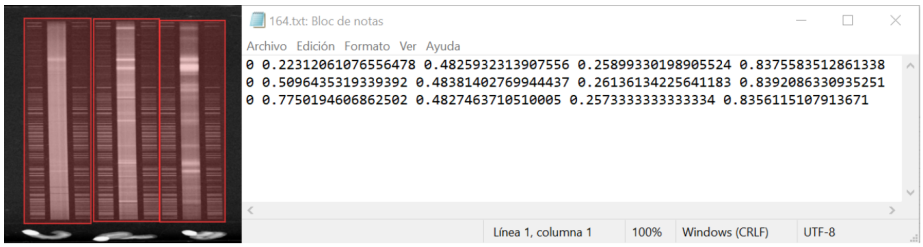

Figura 5.1: a) Imagen del escaneo número 164 con una ilustración de sus respectivas etiquetas. b) Archivo TXT con la información de las etiquetas correspondientes a la imagen de escaneo número 164.

5.4.3. Generación de nuevos elementos

Naturalmente, 256 imágenes es un valor mas bien pequeño para las dimensiones de un *dataset* sobre todo cuando lo que se busca es entrenar una RNC para la detección de objetos. Por tanto, se decidió hacer uso de técnicas de *data augmentation* para aumentar la cantidad de ejemplos disponibles.

La generación de las nuevas imágenes es realizada al momento del entrenamiento. Antes de ello, lo que se debe hacer es especificar que tipo de transformaciones se van realizar y en qué magnitud. En este caso, para la generación de imágenes de placas espectográficas, se utilizaron los siguientes parámetros:

- Cambio de brillo $\pm 30\,\%$
- Ruido gaussiano del $10\,\%$ al $50\,\%$
- Volteo completo de la imagen de forma vertical
- Rotación $\pm 3°$
- Escalado $\pm 20\,\%$
- Mosaico

Naturalmente, estos rangos operacionales para conversiones fueron validados por un profesional del campo astronómico, en pos de asegurar que las nuevas imágenes mantengan el realismo como para seguir siendo consideradas placas espectrográficas.

Evidentemente, para el entrenamiento no basta solo con generar nuevas imágenes, también se debe generar los archivos de etiquetados adecuados para cada una. Para ello lo que se hace es partir del archivo de etiquetas de la imagen original y modificarlo según las transformaciones aplicadas. Naturalmente, la dificultad para hacer estos ajustes varía según las transformaciones que se busque aplicar (no es lo mismo ajustar las etiquetas para un cambio de brillo que para un desplazamiento). Afortunadamente, este proceso es bastante común por lo que las propias librerías ya lo contemplan cuando se generan imágenes de esta forma.

5.4.4. Entrenamiento del modelo

El modelo elegido para resolver el problema fue YOLOv5s, no es el modelo mas preciso de la familia YOLO pero tienen un rendimiento adecuado, además es rápido y entrena de forma mas eficiente que la mayoría de alternativas, por tanto se lo considero adecuado para el problema de detección a tratar. Para el entrenamiento se utilizó un tamaño de lote de 64, un total de 250 épocas y se utilizaron los parámetros de *data augmentation* mencionados en la sección anterior. Únicamente un 80 % de las imágenes fueron utilizadas para el entrenamiento y el 20 % restante se mantuvo para posteriormente realizar el proceso de evaluación. También se redujeron las imágenes de sus tamaños originales a una resolución de 512x512 píxeles con una profundidad de color de 8 bits, se pierde cierta cantidad de información pero de esta forma se homogeneizan las dimensiones de los datos de entrada y se optimiza hasta cierto punto la cantidad de información a procesar.

El entrenamiento se realizó en el entorno de programación con Python Google Colab. Haciendo uso de los recursos de procesamiento que ofrece la versión gratuita de Colab el entrenamiento duró aproximadamente 50 minutos. Esta versión de Colab estipula ciertos límites en los recursos que ofrece según que tan solicitados estén, pero en la oportunidad en que el entrenamiento fue realizado Colab ofrecía aproximadamente 12 GB de RAM, 15 GB de GPU y 70 GB de almacenamiento

Un dato que es de interés para lo que sigue es que YOLO no es un paquete que se instala así como PyTorch o OpenCv, para usarlo es necesario clonar su repositorio ("https://github.com/ultralytics/yolov5") e instalar en el entorno en el que se lo vaya a usar las dependencias que le corresponden. Esto es así tanto para entrenar un modelo YOLO como para utilizar un modelo ya entrenado.

5.4.5. Análisis de rendimiento

Aprovechando las facilidades de la librería PyTorch y los archivos de entrenamiento generados por YOLO se realizaron las correspondientes pruebas sobre el conjunto de evaluación, obteniendo las métricas usuales para este tipo de problema.

La métrica de precisión del modelo resultante fue excelente, con un valor de 0.97, por lo cual se puede tener confianza en que cuando el modelo detecta un espectro es poco probable que lo hallado en realidad no lo sea. La métrica de exhaustividad tomó un valor del 0.99, por lo que también se puede determinar que es muy poco probable que un espectro positivo pase por el modelo sin ser identificado. Realizando la media armónica de estos dos valores se puede calcular la métrica derivada *f-measure*, esta toma el valor de 0.979. En la figura 5.2 se muestra un ejemplo de detecciones realizadas con el modelo antes entrenado. Las predicciones que éste realiza se comportan según lo deseado, ya que de esta muestra se detectaron todos los espectros, a excepción de dos (ver segunda placa de la figura), lo cual está bien, ya que estos dos no son espectros válidos para el análisis por estar corruptos con manchas de humedad.

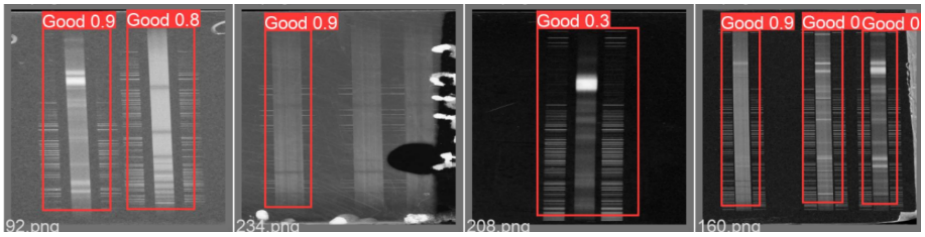

Figura 5.2: Validación del modelo entrenado en algunas imágenes del conjunto de Evaluación.

5.4.6. Despliegue del modelo

Para efectivamente utilizar el modelo en un proyecto primero se debe exportar los pesos obtenidos luego del entrenamiento, estos describen las características del modelo entrenado, por lo que con estos se puede generar y usar el modelo donde sea necesario.

En el caso del Backend el único lugar donde interesa hacer inferencias es en la función correspondiente al *endpoint* "api/predict". Pero en pos de una correcta modularización, se creó una clase de utilidad llamada "Detector" la cual centraliza la lógica necesaria para cargar los pesos del modelo y utilizarlo para inferir la posición de los espectros de las imágenes que reciba. Con esto la función responsable de "api/predict" simplemente organiza la lógica del envío y recepción de parámetros delegando el trabajo de predicción a la clase antes mencionada.

Capítulo 6

Conclusiones

En este trabajo se presenta el desarrollo de un software que asiste durante el proceso de digitalización de placas espectrográficas. El software consta de dos partes principales: un Frontend escrito en Javascript que soluciona la interacción con el usuario y un Backend escrito en Python que resuelve las funcionalidades más complejas. La aplicación resultante, es potencialmente ejecutable en cualquier navegador web.

El software desarrollado incorpora varias herramientas para ayudar al usuario en la mayoría de etapas del proceso de digitalización de placas espectrográficas: utiliza una RNC YOLO para automatizar la detección de espectros en los escaneos, colabora en la obtención de metadatos facilitando al usuario los valores que pueda y automatiza totalmente la exportación y almacenamiento de los archivos resultantes, entre otras funciones. El modelo utilizado para la detección de espectros fue entrenado desde cero en base a datos proveídos por la FCAG y mostró excelentes resultados tanto con datos de validación como en su uso práctico. Para el desarrollo, también se tuvo en cuenta la opinión de astrónomos profesionales en el campo, y con sus aportes se diseñaron y mejoraron muchas funcionalidades que incrementan la usabilidad del software, tales como el guardado automático o el uso de filtros para la visualización de escaneos.

En la práctica, el uso del sistema supuso una mejora sustancial en los tiempos de digitalización, reduciendo el tiempo de procesado estándar de una hora por archivo a aproximadamente 15 minutos. En particular, la FCAG posee una basta colección de placas espectrográficas, siguiendo métodos estándar su procesamiento total tardaría aproximadamente 8 años en completarse (suponiendo que todo lo haga una persona trabajando 8 horas diarias de lunes a viernes). Pero, incorporando el uso del sistema se estima que este tiempo se podría reducir a tan solo 2 años. En definitiva, el software cumple su objetivo principal de reducir el tiempo que se requiere para digitalizar de cada placa espectrográfica, y si se le da el debido soporte, podría contribuir notoriamente en la conservación del patrimonio observacional histórico que las mismas representan.

6.1. Trabajos futuros

Como trabajos futuros respecto a lo aquí presentado se propone:

- **Continuar optimizando la eficiencia de carga**: actualmente, la carga de información por parte del usuario supone el mayor limitante para los tiempos de procesamiento. En pos de bajar aún mas estos tiempos, es necesario explorar cómo las interacciones con el usuario se pueden reducir o hacer mas rápidas. Algunas formas en las que se podría optimizar la eficiencia de carga son: mejorar la organización de la interfaz para que sea mas ágil, mejorar las interfaces de carga para datos espaciales como RA y DEC, integrar el escaneo de los espectros al sistema para que no se tenga que realizar de forma separada, investigar formas de automatizar la carga de aquellos metadatos que actualmente llena el usuario, o también se puede buscar cómo aproximarse a aquellos metadatos que no son determinables para poder darle sugerencias al usuario de que valores les corresponden. Hay varios aspectos de la aplicación que pueden ser investigados y desarrollados, capaz la mejora individual de cada uno no impacte demasiado en el rendimiento de la aplicación, pero acumulando las optimizaciones de cada uno se puede obtener una mejora significativa.
- **Análisis de lámparas de comparación**: aun después de asegurar la conservación de cada espectro, hay que extraer la información de cada uno realizando diversas tareas. La primera es la calibración de las lámparas de comparación. Ésta es una tarea manual y lenta, y aplicarla a los miles de espectros de la FCAG sería inviable a corto plazo. Aun así, esto podría cambiar si se logra automatizarla. Se cree que es viable enfrentarla haciendo uso de AA, por lo que ya se está trabajando en la creación de modelos de AA que la resuelvan.
- **Análisis de espectros**: igual que para el inciso anterior, el procesamiento posterior del espectro es una tarea manual, donde se identifican las frecuencias de interés de un objeto particular. Como paso siguiente a la calibración de lámparas, se espera poder procesar la información existente en las observaciones, obteniendo las diferentes curvas espectroscópicas para cada objeto. De igual forma, automatizar esta tarea contribuiría enormemente a reducir el tiempo requerido para extraer la información útil de cada espectro digitalizado.
- **Incorporación de un sistema de administración de bases de datos (DBMS)**: actualmente, el sistema de almacenamiento utilizado satisface de forma mas que adecuada los requisitos de la aplicación. No obstante, si interesara incorporar un sistema de búsqueda e indexación de espectros entonces el rendimiento de la DB podría verse comprometido, en esos casos sería una buena opción contar con un DBMS que indexe las búsquedas y devuelva los archivos que se requieran un un tiempo mínimo. Dado el tipo de datos que se manejan, un DBMS orientado a documentos como MongoDB podría ser una buena opción.

- **Aceptar múltiples usuarios**: el sistema actualmente está pensado para que cada instancia atienda a un único usuario, pero en pos de optimizar el uso de recursos podría ser de interés adaptar el Backend para que acepte solicitudes de múltiples usuarios simultáneamente. En caso de hacerlo, la centralización del almacenamiento en el Backend permitirá que las modificaciones a realizar sean mínimas, pero se deberá tomar decisiones respecto a qué ocurrirá cuando dos usuarios intenten trabajar sobre la misma placa u otras situaciones similares. Además, sería conveniente examinar la capacidad de respuesta del Backend ante gran cantidad de solicitudes.

Bibliografía

[Bernabé-Mónica, 2016] Bernabé-Mónica (2016). La cámara de las maravillas — cultura — el mundo. `https://www.elmundo.es/cultura/2016/05/03/57221fba46163f7f4d8b4577.html`.

[Bernaola, 2004] Bernaola, O. A. (2004). Enrique gaviola and the astronomical observatory of córdoba. *Astrophysics and Space Science*, 290(3):457–461.

[DASCH, 2017] DASCH (2017). Dasch: Digital access to a sky century @ harvard a new look at the temporal universe. `http://dasch.rc.fas.harvard.edu/project.php`.

[Detlef et al., 2014] Detlef, G., Taavi, T., Heinz, E., Rainer, A., Uli, H., and Harry, E. (2014). *Building up APPLAUSE: Digitization and web presentation of Hamburger Sternwarte plate archives.*

[Enríquez et al., 2005] Enríquez, D. G., Soler, E. M., García, V. J. M., and Torres, J. A. M. (2005). *Astronomía fundamental*, volume 81. Universitat de Valencia.

[IBMCloudEducation, 2020] IBMCloudEducation (2020). ¿qué son las redes neuronales? - argentina — ibm.

[Joye and Mandel, 2005] Joye, W. and Mandel, E. (2005). The development of saoimage ds9: Lessons learned from a small but successful software project. In *Astronomical Data Analysis Software and Systems XIV*, volume 347, page 110.

[Landup, 2022] Landup, D. (2022). Randaugment for image classification with keras/tensorflow. (Accessed on 02/09/2023).

[Mandel et al., 2007] Mandel, H., Birkle, K., Demleitner, M., and Heidelberg, L. (2007). HDAP – heidelberg digitized astronomical plates. VO resource provided by the GAVO Data Center.

[Mayer-Schönberger and Cukier, 2013] Mayer-Schönberger, V. and Cukier, K. (2013). *Big data: la revolución de los datos masivos.* Turner.

[Meilan, 2020] Meilan, N. (2020). Recuperando nuestro pasado astronomico. *SI MUOVE*, pages 31–36.

[Meilán, 2018] Meilán, N. S. (2018). *Recuperación del patrimonio observacional histórico*. PhD thesis, Universidad Nacional de La Plata.

[Mitchell, 1995] Mitchell, M. (1913-1995). Astronomical plates — maria mitchell association. `https://www.mariamitchell.org/astronomical-plates-collection`.

[Muminov et al., 2017] Muminov, M., Yuldoshev, Q., Ehgamberdiev, S., Kahharov, B., Relke, H., Protsyuk, Y., Pakuliak, L., and Andruk, V. (2017). Astrometry of the h and χ persei clusters based on the processing of digitized photographic plates. *Bulg. Astron. J*, 26:3–14.

[Oster, 2021] Oster, L. (2021). *Astronomía moderna*. Reverté.

[Paolantonio, 2021a] Paolantonio, S. (2021a). Eclipses de Sol observados en la República Argentina: período 1810 - 1950. *Boletin de la Asociacion Argentina de Astronomia La Plata Argentina*, 62:304–306.

[Paolantonio, 2021b] Paolantonio, S. (2021b). Observaciones astronómicas en la República Argentina antes del Observatorio Nacional. *Boletin de la Asociacion Argentina de Astronomia La Plata Argentina*, 62:298–300.

[Redmon et al., 2015] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640.

[ReTrhOH, 2019] ReTrhOH (2019). Retrhoh — recuperación del trabajo observacional histórico. `https://retroh.fcaglp.unlp.edu.ar/astronomia/`.

[Rieznik, 2013] Rieznik, M. (2013). The córdoba observatory and the history of the 'personal equation' (1871–1886). *Journal for the History of Astronomy*, 44(3):277–301.

[Schroeder et al., 2020] Schroeder, A. B., Dobson, E. T., Rueden, C. T., Tomancak, P., Jug, F., and Eliceiri, K. W. (2020). The imagej ecosystem: Open-source software for image visualization, processing, and analysis. *Protein Science*, 30(1):234–249.

[Skalski, 2019] Skalski, P. (2019). Releases · skalskip/make-sense. `https://github.com/SkalskiP/make-sense`.

[Taavi et al., 2014] Taavi, T., Heinz, E., Detlef, G., and Harry, E. (2014). *Work flow for plate digitization, data extraction and publication*.

[Tkachenko et al., 2022] Tkachenko, M., Malyuk, M., Holmanyuk, A., and Liubimov, N. (2020-2022). Label Studio: Data labeling software. Open source software available from https://github.com/heartexlabs/label-studio.

[UAI, 2000] UAI (2000). Nro. b3 "safeguarding the information in photographic plates". `https://www.iau.org/static/publications/ib88.pdf`.

[Walborn et al., 2017] Walborn, N. R., Gamen, R. C., Morrell, N. I., Barbá, R. H., Lajús, E. F., and Angeloni, R. (2017). Active luminous blue variables in the large magellanic cloud. *The Astronomical Journal*, 154(1):15.

[Wells et al., 1981] Wells, D. C., Greisen, E. W., and Harten, R. H. (1981). FITS - a Flexible Image Transport System. *AAPS*, 44:363.

[Wenger, M. et al., 2000] Wenger, M., Ochsenbein, F., Egret, D., Dubois, P., Bonnarel, F., Borde, S., Genova, F., Jasniewicz, G., Laloë, S., Lesteven, S., and Monier, R. (2000). The simbad astronomical database - the cds reference database for astronomical objects. *Astron. Astrophys. Suppl. Ser.*, 143(1):9–22.

[Wertz et al., 2017] Wertz, M., Horns, D., Groote, D., Tuvikene, T., Czesla, S., and Schmitt, J. H. M. M. (2017). Hamburger sternwarte plate archives: Historic long-term variability study of active galaxies based on digitized photographic plates. *Astronomische Nachrichten*, 338(1):103–110.

Apéndice A

API Backend

La API ofrecida por el Backend esta conformada por 7 *endpoint*. El presenté listado muestra la informacion necesaria para usar cada uno:

1. "/api/allPaths":
   - Método: GET
   - Al ejecutar: Informa un listado de todos los archivos TIFF que haya en la carpeta especificada.
   - Parámetros:
     - **path_dir**: Path de almacenamiento donde se quiere consultar.
   - Body: -
   - Respuesta Exitosa: 200 OK:

```
{
    "paths": [
        {
            "fileName": "1653318964.tiff",
            "number_of_spectra": 0,
            "saved": false
        },
        {
            "fileName": "1653319500.tiff",
            "number_of_spectra": 4,
            "saved": true
        },
        ...
    ]
}
```

2. "/api/loadConfig":
   - Método: GET

- Al ejecutar: Informa varios datos útiles para cargar correctamente la aplicación. Actualmente, informa el listado de observatorios a mostrar en el campo OBSERVAT y el Path de donde el backend está buscando los archivos TIFF.
- Parámetros: -
- Body: -
- Respuesta Exitosa: 200 OK:

```
{
    "config": {
        "fields": {
            "OBSERVAT": {
                "options": [
                    "ctio: Cerro Tololo Interamerican
                    Observatory",
                    "kpno: Kitt Peak National
                    Observatory",
                    ...
                ]
            }
        },
        "global": {
            "workspace_path": "../test_db/"
        }
    }
}
```

3. "/api/image/load":

- Método: GET
- Al ejecutar: Permite cargar los datos de una imagen especificada, en un determinado path de almacenamiento.
- Parámetros:
  - **dir_path**: Path de almacenamiento donde se quiere consultar.
  - **img_name**: Nombre de la imagen de la que se quieren sus datos.
- Body: -
- Respuesta Exitosa: 200 OK:

```
{
    "image": "iVBOR...(imagen en base64)",
    "info": {
        "dtype": "uint16",
        "ext": "tiff",
        "heigth": 673,
        "name": "4405",
```

```
            "path": "../test_db/4405.tiff",
            "width": 2844,
            # Si se tiene información guardada del
            # procesamiento de una imagen entonces
            # se incluye a través de los siguientes
            # campos:
            "plateData": {
                "DIGITALI": "Thompson",
                ...
            },
            "metadata": [
                {
                    "AIRMASS": "",
                    ...
                },
                ...
            ],
            "bboxes": [
                {
                    "h": 221.716796875,
                    "w": 2675.1484375,
                    "x": 197.50987499999997,
                    "y": 51.83120964229937
                },
                ...
            ],
        },
        "status": true
    }
```

4. "/api/image/save":

   - Método: PUT
   - Al ejecutar: Permite guardar el valores de los metadatos y BBs de una imagen en un momento determinado de su procesamiento.
   - Parámetros: -
   - Body:

     ```
     {
         "img_name":"4405.tiff",
         "img_path":"../test_db/"
     }
     ```

   - Respuesta Exitosa: 200 OK:

     ```
     {
         "predictions": [
     ```

```
            {
                "h": 0.591796875,
                "w": 0.91015625,
                "x": 0.478515625,
                "y": 0.5322265625
            },
            ...
        ]
    }
```

5. "/api/getMetadata":
   - Método: POST
   - Al ejecutar: Calcula todos los metadatos que se puedan calcular en base a los datos proveídos y retorna el listado resultante al usuario (Mas detalles respecto a su funcionamiento en la sección 5.3).
   - Parámetros: -
   - Body:

```
{
    "OBJECT":"N2",
    "OBSERVAT":"ctio: Cerro Tololo Interamerican
    Observatory",
    "DATE-OBS":"1980-02-10",
    "UT":"12:05:22"
    # Con esos 4 metadatos puede calcular la
    # cantidad maxima de metadatos derivatos. Si no
    # llegase a tener el valor de alguno puede
    # enviarlo con el valor "MISSING"
}
```

   - Respuesta Exitosa: 200 OK:

```
{
    "message": "success",
    "metadata": {
        "AIRMASS": "1.30887",
        "DATE-OBS": "1980-02-10",
        ...
    }
}
```

6. "/api/predict":
   - Método: POST
   - Al ejecutar: Genera predicciones en base a la imagen especificada respecto a la cantidad de Bounding Boxes, sus dimensiones y posición (Mas detalles respecto a su funcionamiento en la sección 5.4).

- Parámetros: -
- Body:

```
{
    "img_name":"4405.tiff",
    "img_path":"../test_db/"
}
```

- Respuesta Exitosa: 200 OK:

```
{
    "predictions": [
        {
            "h": 0.591796875,
            "w": 0.91015625,
            "x": 0.478515625,
            "y": 0.5322265625
        },
        ...
    ]
}
```

7. "/api/generatefits":

- Método: POST
- Al ejecutar: Genera los archivos FITS correspondientes a la información recibida y los guarda en la db.
- Parámetros: -
- Body:

```
{
    "img_name":"4405.tiff",
    "path_dir":"../test_db/",
    "bbox_arr":[
        {
            "x": 197.50987499999997,
            "y": 51.83120964229937,
            "h": 221.716796875,
            "w": 2675.1484375
        },
        ...
    ],
    "data_arr": [
        {
            "AIRMASS": "1.30887",
            "DATE-OBS": "1980-02-10",
            ...
```

```
            }
        ],
        "plate_data": {
            "DIGITALI": "Thompson",
            "OBSERVAT": "ctio: Cerro Tololo
            Interamerican Observatory",
            ...
        },
        "fields": {
            "object_": {
                "label": "OBJECT",
                "type": "text",
                "info": "Name of the object observed",
                "required": true,
                "global": false
            },
            ...
        }
}
```

- Respuesta Exitosa: 200 OK:

```
{
    "status": True
}
```