UNIVERSIDADE DE SANTIAGO DE COMPOSTELA


ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

# IDALib:
# Desarrollo de una librería en Python para *Image Data Augmentation*

*Autora:*
**Raquel Rodríguez Vilas**

*Tutores:*
**María José Carreira Nouche**
**Nicolás Vila Blanco**

**Grado en Ingeniería Informática**

**Julio 2020**

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática

**Dña. María José Carreira Nouche**, Profesora del Departamento de Electrónica y Computación de la Universidade de Santiago de Compostela, y **D. Nicolás Vila Blanco**, Investigador del Departamento de Electrónica y Computación de la Universidade de Santiago de Compostela,

INFORMAN:

Que la presente memoria, titulada *IDALib: Desarrollo de una librería en Python para Image Data Augmentation*, presentada por **Dña. Raquel Rodríguez Vilas** para superar los créditos correspondientes al Trabajo de Fin de Grado de la titulación de Grado en Ingeniería Informática, se realizó bajo nuestra tutorización en el Departamento de Electrónica y Computación de la Universidade de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a (Data):

| La tutora, | El cotutor, | La alumna, |
|---|---|---|
| CARREIRA NOUCHE MARIA JOSE - 33272482S 2020.07.02 17:23:07 +02'00' | VILA BLANCO NICOLAS - 45872236R Firmado digitalmente por VILA BLANCO NICOLAS - 45872236R Fecha: 2020.07.02 15:27:58 +02'00' | |
| María José Carreira Nouche | Nicolás Vila Blanco | Raquel Rodríguez Vilas |

# Agradecimientos

En especial a María José y Nico, por la confianza, ayuda y apoyo que me habéis aportado desde el primer momento.

A mi hermano, por todas las veces que me has instalado Linux. A mi hermana, por cuidar de mi y de mi salud mental. Y a mi madre, por todo lo demás.

A mis compañeras de piso, por las cervezas, risas, convivencia y por escoger siempre tan bien los pisos. Santiago no habría sido lo mismo sin vosotras.

A mis amigos de clase, y a Dabo y Alex sois lo mejor que me dio Santiago. Sé que sin vosotros llegar hasta aquí no habría sido posible.

Y por supuesto a Yors, por acompañarme estos años y nunca dejar de apoyarme.

# Resumen

En esta memoria se refleja el trabajo realizado para la construcción de la librería en Python de *image data augmentation* **IDALib**. Como se detalla en esta memoria, la tarea de *image data augmentation* es un procedimiento asociado al auge de la inteligencia artificial y requerimientos de las redes neuronales artificiales, en concreto a aquellas que trabajan con imágenes y otros datos multidimensionales.

El desarrollo de esta herramienta quiere ser un granito de arena más en la imparable evolución del mundo de la Inteligencia Artificial:

> *Al igual que la electricidad transformó casi todo hace un siglo, hoy en día se me hace difícil imaginar una industria que no sea transformada por la Inteligencia Artificial en los próximos años.* – **Andrew Ng**

# Índice general

# Índice de figuras

# Índice de cuadros

# Capítulo 1

# Introducción

En la actualidad la inteligencia artificial y en concreto el aprendizaje profundo (*deep learning*) está liderando un nuevo sector de desarrollo tecnológico que aborda un sinfín de oportunidades.

En particular, el tratamiento de imágenes con redes neuronales convolucionales supone un área muy extensa y con aplicaciones que albergan desde la detección de objetos en coches inteligentes o el reconocimiento facial en redes sociales hasta el análisis de imágenes médicas en el mundo de la e-salud.

## 1.1. Motivación

El desarrollo de este tipo de redes incorpora una tarea de entrenamiento que requiere de una gran cantidad de datos de entrada (en este caso de imágenes) que no siempre está a nuestra disposición. Esta necesidad de grandes cantidades de datos es lo que conduce al concepto de *image data augmentation* o aumento de datos de imágenes, una técnica para aumentar el tamaño de los conjuntos de datos de entrada a partir de pequeñas transformaciones de las imágenes originales.

Entre estas transformaciones se incluyen traslaciones de la imagen, rotaciones, cambios en el brillo, en el contraste... que aplicadas de manera cuidadosa transformarán las imágenes de partida en *nuevas imágenes* para la red neuronal, permitiéndonos llevar a cabo un entrenamiento completo y mejorado de la red.

El proceso de aumento de datos de imágenes (*image data augmentation*) se ilustra en la Figura 1, donde a partir de una única imagen se generan nuevas imágenes con ligeras variaciones sobre la original. Estas nuevas imágenes conservan las características que deben servir a una red neuronal para identificar en este caso, al animal representado en la imagen. No obstante, la imagen como mapa de bits ha sufrido cambios lo suficientemente notables para que sea interpretada como una nueva imagen para el entrenamiento de la red.

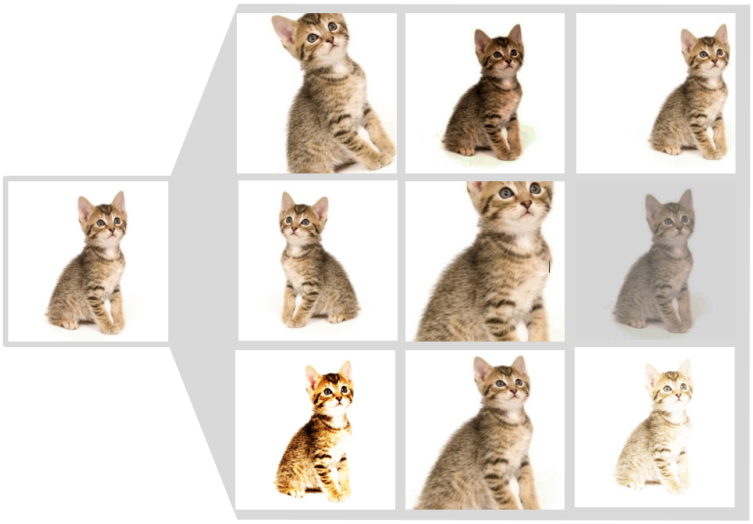


Figura 1.1: Transformaciones de una imagen para aumentar el conjunto de imágenes de entrada.

Este proceso no solo nos permite obtener un mayor número de elementos de entrada, sino que proporciona un aprendizaje más robusto, independiente de la posición, color o tamaño del elemento a identificar.

El entrenamiento de una red neuronal es una tarea increíblemente costosa a nivel de procesamiento, (por lo general el entrenamiento de redes neuronales puede durar días), por lo que cualquier tarea adicional que se le sume a este proceso debe ser lo más eficiente posible para evitar una sobrecarga mayor que no sea totalmente esencial. La principal motivación de este trabajo es el desarrollo de una librería para la realización de la tarea de *image data augmentation* de la forma más eficiente posible para así centrar los esfuerzos de procesamiento en el entrenamiento de la red.

## 1.2. Objetivos generales

El objetivo de este proyecto es el desarrollo de una librería en Python que realice las operaciones de *image data augmentation* con el propósito de:

1. Lograr la máxima eficiencia computacional.

2. Orientarla a su integración con redes neuronales.

Hay que tener en cuenta que los *datasets* de entrada para el entrenamiento de la red no siempre se componen únicamente de imágenes o datos individuales, sino que cada una de estas imágenes o datos de entrada puede acompañarse de los

datos que deseamos que aprenda a identificar la red en ella. Por ejemplo: si queremos que identifique determinados puntos característicos de una imagen, la entrada para que nuestra red "aprenda" estará formada por las imágenes originales acompañadas de las coordenadas de los puntos que "marcan" estas características. Es decir, se acompaña cada dato de entrada con la salida que debería generar la red una vez entrenada. Por lo tanto, el proceso de *image data augmentation* para el entrenamiento de la red debería transformar también estos metadatos o "marcas".

Como se ve en la Figura 1.2 estos metadatos pueden ser de varios tipos:

- **Máscaras**: mapas de bits compuestos por valores binarios (1 o 0) empleados para marcar cierta zona de la imagen (como puede ser marcar la zona de la imagen en la que hay un coche).
- **Mapas de segmentación:** (*segmap*) generalización de las máscaras. Pueden tomar más de dos valores para "marcar"más de una zona diferenciada en la imagen. Por ejemplo separar la sección de imagen ocupada por carrocería, la ocupada por las ruedas, la ocupada por el suelo...
- **Mapa de calor**: mapa de bits enfocado a "marcar"zonas de la imagen pudiendo diferenciar mayores o menores valores por sección, son útiles para indicar probabilidad.
- **Puntos característicos**: (*keypoints*) coordenadas de puntos que marcan cierta característica o características concretas de la imagen.

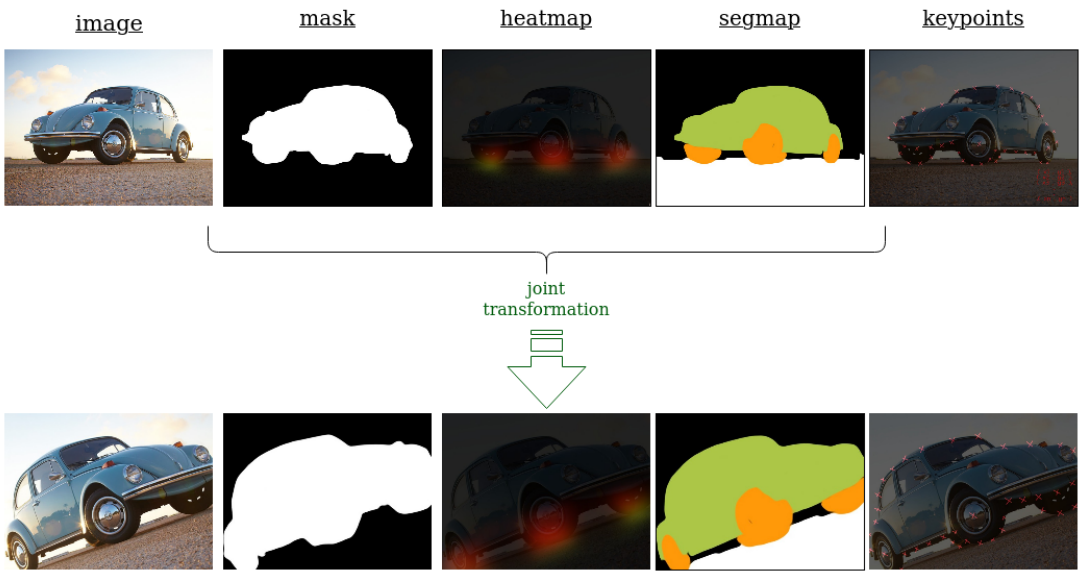



Figura 1.2: Ejemplo de distintos tipos de metadatos y su transformación.

Por todo lo anterior, y a diferencia de otras herramientas ya existentes para *image data augmentation*, además de incorporar las operaciones propias sobre las imágenes originales, se incluirán las transformaciones conjuntas sobre sus metadatos asociados. Es decir, la librería a desarrollar debe permitir realizar operaciones de aumento de datos sobre las imágenes y sus metadatos (coordenadas de puntos, mapas de saliencia o calor o máscaras relativas al dato original) de manera que se mantenga la relación original entre estas anotaciones y la entrada a transformar en toda operación, como se muestra en la Figura 1.2.

### 1.2.1. Subobjetivos

Para alcanzar estas metas, en el proyecto se considerarán los siguientes subobjetivos:

1. Estudio individual de cada transformación para determinar la implementación más eficiente, considerando la paralelización de tareas o su realización en GPU o CPU.

2. Implementación eficiente de cada operación de transformación.

3. Desarrollo de un *pipeline* que combine de manera eficiente conjuntos de transformaciones.

4. Desarrollo de un sistema eficiente de alimentación para redes neuronales que construya los lotes de elementos de entrada para una red neuronal (integrando el *image data augmentation*) de manera ligera para el sistema.

5. Desarrollo de un sistema de generación del *dataset* aumentado a disco.

6. Desarrollo de un sistema de visualización de las transformaciones que facilite la depuración de los programas que hagan uso de la librería.

7. Integración y compactación de las operaciones implementadas en formato de librería.

## 1.3. Glosario

En este apartado se definen aquellos términos empleados a lo largo de la memoria y que son específicos del campo de aplicación de la misma:

**Image data augmentation** : técnica que se puede utilizar para expandir artificialmente el tamaño de un conjunto de datos de entrenamiento creando versiones modificadas de imágenes en el conjunto de datos. [1]

[1] Se emplea el término inglés al no existir una traducción directa y ser el nombre estándar para denominar dicha tarea.

**API** : interfaz de programación para aplicaciones. Se entiende por API a los procesos, las funciones y los métodos que brinda una determinada biblioteca de programación para que sea empleada por otra aplicación.

***Pipeline*** : término que en el ámbito de la programación se usa para hacer referencia a una arquitectura que consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior.

**Tensor** : tipo de estructura de datos que se utiliza en el álgebra lineal y que es una generalización de matrices, pudiendo entenderse como una matriz $n$-dimensional.

**CUDA** (Compute Unified Device Architecture): módulos NVIDIA de computación paralela eficiente.

**NumPy** : paquete estándar de Python de soporte y tratamiento de vectores y matrices multidimensionales.

**Torch tensor** : tipo de dato de la librería Pytorch empleado para expresar tensores.

**Dataloader** : estructura de datos de Pytorch que gestiona los elementos de un *dataset*, organizando el orden y conjuntos de imágenes.

**Profundidad de color** : número de bits destinados a la codificación del color de cada píxel. Con una profundidad de color de 8 bits, se pueden codificar 256 colores diferentes.

**IDE** (Integrated Development Environment): aplicación informática que proporciona servicios para facilitar el desarrollo de software.

**Benchmark** : también llamado prueba de rendimiento o comparativa, es una técnica utilizada en informática para medir la calidad o el rendimiento de un sistema o uno de sus componentes.

**GPU** : coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central.

**Red neuronal** : en esta memoria, hace referencia a las redes neuronales artificiales. Modelo de computación inspirado en el funcionamiento del cerebro humano que trata de .ªprenderün determinado comportamiento.

**PyPI** (Python Package Index): repositorio de software oficial para aplicaciones de terceros en el lenguaje de programación Python.

## 1.4. Estructura de la memoria

La presente memoria está dividida en 7 capítulos que reflejan, desde diferentes perspectivas, el trabajo desarrollado:

**Capítulo 1- Introducción:** en este capítulo se lleva a cabo una breve introducción del TFG realizado, detallando su motivación y objetivos.

**Capítulo 2- Estado del arte:** en este capítulo se expone la situación actual de las herramientas y técnicas disponibles. Se describe la propuesta de librería así como las mejoras que aporta al sector.

**Capítulo 3- Gestión del proyecto:** en este capítulo se detallan todas las tareas relacionadas con la gestión de proyectos llevadas a cabo para la realización del TFG.

**Capítulo 4- Análisis:** capítulo en que se detallan las tareas de análisis llevadas a cabo para el proyecto. En concreto se definen sus funcionalidades mediante el *Product Backlog* y el análisis y selección de tecnologías.

**Capítulo 5- Diseño e implementación:** en este capítulo se exponen las fases de desarrollo del producto, destacando las decisiones de diseño e implementación más relevantes.

**Capítulo 6- Verificación y validación:** en este capítulo, se detallan las pruebas realizadas para verificar el correcto funcionamiento del sistema y el cumplimiento de todos los requisitos establecidos.

**Capítulo 7- Conclusión y posibles ampliaciones:** en este capítulo, se recogen las conclusiones obtenidas tras la finalización del desarrollo del proyecto y se lleva a cabo una reflexión sobre sus posibles mejoras y trabajo futuro.

# Capítulo 2

# Estado del arte

En este capítulo se expone la situación actual de las herramientas, técnicas y modalidades asociadas a la tarea de *image data augmentation*.

En particular, se analizan desde una perspectiva teórica las técnicas para llevar a cabo las transformaciones sobre las imágenes, se estudian las principales herramientas disponibles (analizando sus ventajas y desventajas) para finalizar con la propuesta de librería y la descripción de las necesidades que cubre y mejora respecto a las herramientas actuales.

## 2.1. Transformaciones de imágenes

Como el proyecto está limitado a transformaciones sobre imágenes y datos bidimensionales, nos centraremos en las transformaciones 2D. Diferenciaremos entre transformaciones espaciales o geométricas y transformaciones a nivel de píxel:

- **Transformaciones espaciales (geométricas)**: son aquellas cuya aplicación se realiza sobre la posición de cada píxel. Estas transformaciones modifican de manera conjunta la imagen y objetos que pueda tener asociados como máscaras o coordenadas de puntos. A su vez se subdividen en:

    1. **Transformación isométrica**: cambios de posición (orientación) de una figura determinada que no alteran la forma ni el tamaño de ésta (traslación, *flip* y rotación).
    2. **Transformación de similitud**: transformaciones en las que permanece invariante el ángulo y el paralelismo entre rectas.
    3. **Transformación afín:** transformación que preserva la colinealidad, el paralelismo y ratio de distancias entre puntos.
    4. **Transformación proyectiva**: representa la proyección de la imagen sobre un plano. En esta operación se mantiene la colinealidad.

- **Transformaciones a nivel de píxel**: afectan sólo a los valores de la imagen de entrada, dejando inalterados los objetos adicionales como máscaras o coordenadas de puntos. A su vez pueden ser:

  1. **Transformaciones puntuales**: el valor del píxel de salida depende únicamente del valor del píxel de entrada.
  2. **Transformaciones locales**: el valor del píxel de salida se obtiene teniendo en cuenta el valor de los píxeles vecinos.
  3. **Transformaciones globales**: para calcular el valor del píxel de salida se tienen en cuenta datos de todos los píxeles de la imagen de entrada.

### 2.1.1. Transformaciones espaciales

Las transformaciones espaciales[1] son operaciones que convierten las coordenadas de la imagen de entrada a unas nuevas coordenadas en la imagen de salida usando una función de transformación. Una vez calculadas las nuevas coordenadas, los valores de cada píxel de la imagen de entrada se mapean a los píxeles de la imagen de salida (o viceversa) usando funciones de interpolación (que se explican más adelante).

Para la implementación eficiente de las transformaciones espaciales se emplean las **coordenadas homogéneas o coordenadas proyectivas** [7], que se basan en agregar una dimensión al espacio de trabajo para poder representar las transformaciones mediante matrices:

$$X = \begin{pmatrix} x \\ y \end{pmatrix} \quad \longrightarrow \quad X_h = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.1}$$

El uso de matrices permite calcular las coordenadas de cada píxel de salida mediante una multiplicación matricial:

$$\begin{gathered} \boldsymbol{p_t} = \boldsymbol{T(t)} \cdot \boldsymbol{p} \\ \downarrow \\ \begin{pmatrix} x_t \\ y_t \\ z_t \end{pmatrix} = \begin{pmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,1} & t_{3,2} & t_{3,3} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \end{gathered} \tag{2.2}$$

Aplicando esta multiplicación a cada uno de las coordenadas de los píxeles de entrada, se obtiene la posición que le corresponde a cada píxel en la imagen transformada. Según la operación que se desee realizar, la matriz de transformación tendrá una forma determinada:

[1] A lo largo del proyecto se hace referencia a estas transformaciones por el nombre de transformaciones geométricas o transformaciones espaciales indistintamente.

- **Traslación**:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix} \tag{2.3}$$

$t_x$ = número de píxeles de traslación en el eje X
$t_y$ = número de píxeles de traslación en el eje Y

- **Rotación antihoraria**

$$\begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

$\alpha$ = grados de rotación

- **Escalado**

$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

$S_x$ = factor de escalado horizontal
$S_y$ = factor de escalado vertical

- ***Flip*** **(reflexión) horizontal**

$$\begin{pmatrix} -1 & 0 & W \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

$W$ = ancho de la imagen en píxeles

- ***Flip*** **(reflexión) vertical**

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & H \\ 0 & 0 & 1 \end{pmatrix} \tag{2.7}$$

$H$ = alto de la imagen en píxeles

- **Estiramiento (*shear*)**

$$\begin{pmatrix} 1 & K_x & 0 \\ K_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

$K_x$ = factor de estiramiento horizontal
$K_y$ = factor de estiramiento vertical

La mayor ventaja del uso de matrices es que admiten la **composición de transformaciones**. En el caso de, por ejemplo, querer aplicar a una imagen una operación de rotación, seguida de una operación de escalado y de una de *flip* horizontal (sin componer las operaciones), sería necesario calcular (mediante multiplicación matricial) la posición de cada píxel tras cada operación y realizar la interpolación también después de cada operación. Este procedimiento es muy ineficiente ya que:

- Las multiplicaciones matriciales son muy costosas en el procesado digital de imágenes.
- La operación de interpolación se repite para cada transformación.
- La interpolación implica pérdida de datos y precisión.
- La aplicación secuencial de las transformaciones se convierte en un procedimiento muy costoso, poco escalable y que implica pérdida de precisión.

La composición de transformaciones permite mejorar este procedimiento calculando la multiplicación matricial de las matrices de transformación necesarias, obteniendo una matriz que sintetiza la transformación compuesta. De esta manera, las coordenadas de los píxeles de la imagen original se multiplican una única vez, y se realiza únicamente una operación de interpolación.

Para realizar de manera correcta la composición de transformaciones manteniendo su orden de aplicación, la matriz compuesta de las operaciones se obtiene multiplicando en orden inverso las matrices individuales. Por ejemplo, si aplicásemos rotación (R), después escalado (S) y finalmente simetría horizontal (HF), la matriz compuesta (C) se calcularía como:

$$C = HF \times S \times R \tag{2.9}$$

ya que en multiplicación matricial, la primera transformación en aplicarse es la correspondiente a la matriz que ocupe la posición colocada en el extremo derecho de la expresión.

Otro aspecto a evaluar de las transformaciones es la **interpolación**, que es el proceso mediante el cual se estiman los valores de intensidad de cada píxel de

la imagen de salida a partir de los valores intermedios calculados en la transformación. Se deben usar funciones de interpolación debido a que las coordenadas son valores discretos, mientras que las coordenadas obtenidas por la función de transformación son continuas y pueden generar valores intermedios.

Existen 2 enfoques principales en lo que refiere a la interpolación. Dada una función de transformación ($T$), un píxel de entrada ($x$) y el correspondiente valor del píxel de salida ($x_{output}$):

- **Mapeo hacia adelante (*forward mapping*)**: se calcula la posición que le correspondería a cada píxel de entrada en la imagen de salida. El valor de cada píxel de salida se interpola entre los píxeles de entrada cuya posición transformada los sitúe en la imagen de salida como los más cercanos.

$$x_{output} = T \cdot x \tag{2.10}$$

- **Mapeo hacia atrás (*backward mapping*)**: se aplica la transformación inversa para calcular la posición que le correspondería a cada píxel de la imagen de salida en la imagen original. Su valor se calcula interpolando los píxeles originales más cercanos.

$$x = T^{-1} \cdot x_{output} \tag{2.11}$$

Dependiendo de la transformación que apliquemos, puede ser más eficiente el uso de una u otra aproximación. El mapeo hacia adelante implica calcular la posición de todos los píxeles de entrada cuando, por ejemplo al hacer zoom sobre una imagen, muchos de ellos pueden situarse fuera de la imagen de salida y no ser necesarios, además de que varios puntos de entrada podrían caer en la misma posición de salida. No obstante puede suceder lo mismo con el mapeo hacia atrás si en lugar de zoom se hace un escalado reduciendo el tamaño de la imagen.

Existen tres métodos principales de interpolación:

1. **Vecino más cercano:** es el método computacionalmente más económico. Se basa en copiar el valor del píxel más cercano.

2. **Bilineal:** se promedia linealmente a lo largo de cada fila (eje X) de una imagen, y posteriormente el resultado se interpola linealmente en la dirección por columnas (eje Y).

3. **Bicúbica:** consiste en promediar los valores de los 16 píxeles vecinos. Emplea polinomios de Lagrange, splines cúbicas o bien convolución cúbica para determinar la influencia de cada píxel en el valor final. El resultado obtenido es más preciso que los casos anteriores, pero computacionalmente es más costoso.

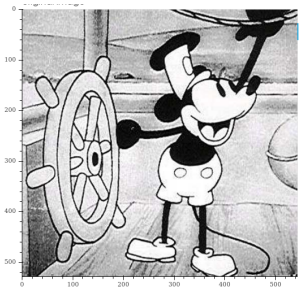
Otro aspecto a abordar en las transformaciones espaciales es qué hacer con aquellos píxeles de la imagen de salida que no se corresponden con ningún píxel de la imagen de entrada, proceso conocido como **extrapolación**. Por ejemplo, si se realiza un escalado a la mitad de tamaño, la mitad de los píxeles de la imagen de salida se situarán *fuera* de los límites de la imagen de entrada.

Existen soluciones muy diversas para estos píxeles, y en la Figura 2.1 mostramos las tres más comunes:

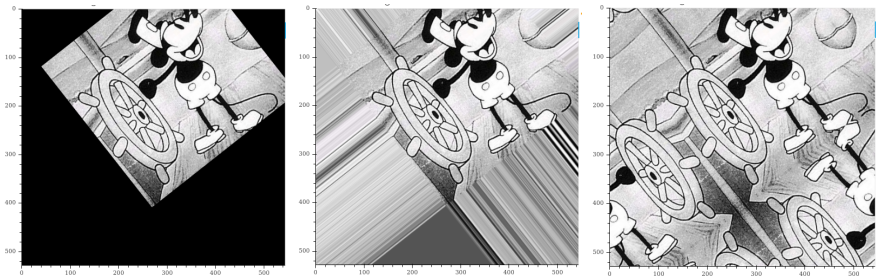
1. **Rellenado de ceros (*zero padding*)**: todos los píxeles que correspondan a posiciones "fuera"de la imagen original tendrán valor cero.

2. **Reflexión (*reflection padding*)**: todos los píxeles que correspondan a posiciones "fuera"de la imagen original se rellenan reflejando los valores de entrada a través del eje del límite de la imagen.

3. **Réplica (*replication padding*)**: todos los píxeles que correspondan a posiciones "fuera"de la imagen original replican los valores del límite más cercano.



(a) Imagen original



(b) Rellenado de ceros (c) Reflexión (d) Réplica

Figura 2.1: Ejemplo de aplicación de los tipos de extrapolación más comunes.

### 2.1.2. Transformaciones a nivel de píxel

Por lo general, son las transformaciones espaciales las que requieren de un proceso más costoso, además de afectar a todos los elementos bidimensionales

que puedan acompañar a la imagen. No obstante, las transformaciones a nivel de píxel[2] ofrecen una variabilidad en las imágenes de gran valor para el *image data augmentation* y, por lo general, por un coste computacional menor.

Existe gran diversidad entre este tipo de operaciones, aunque aquí realizaremos un análisis de las más comunes dentro del contexto del trabajo:

**Transformaciones puntuales**

- **Cambio de brillo:** modificación del valor de la intensidad de cada píxel. La fórmula más sencilla es:

$$p_t = p + brightness \tag{2.12}$$

- **Cambio de contraste:** se puede entender el contraste como la diferencia entre el valor máximo y mínimo de intensidad de los píxeles de una imagen. Para cambiar el contraste de una imagen se emplea la fórmula [3]:

$$p_t = contrast \times (p - 128) + 128 \tag{2.13}$$

- **Corrección gamma:** operación no lineal que se usa para codificar y decodificar luminancia. Para llevar a cabo la corrección gamma de una imagen se emplea la fórmula[3] :

$$p_t = 255 * \left(\frac{p}{255}\right)^{\frac{1}{gamma}} \tag{2.14}$$

La ventaja de estas operaciones puntuales es que, como el valor del píxel de salida solo depende del valor del píxel de entrada, permiten la composición de transformaciones. Las funciones de transformación como las expresadas en las ecuaciones (2.12), (2.13) o (2.14) son operaciones matemáticas que admiten composición, por lo que es posible sintetizar múltiples operaciones en una única función de transformación.

Otra ventaja interesante al trabajar con estas transformaciones son las ***LookUp Tables* o LUTs**, una estructura de datos (por lo general un vector) que permite hacer una asociación directa entre el valor de entrada de un píxel y el valor que le corresponde tras la transformación. El uso de esta herramienta permite calcular una única vez el valor que le correspondería a cada valor de entrada (para construir la tabla) y, una vez construida, el procesado de dicha transformación presenta una complejidad computacional $\Theta(n)$ siendo n el número de píxeles de la imagen (linealmente proporcional). En general el uso de LUTs aumenta la velocidad de procesdo y reduce la sobrecarga de estos métodos.

[2]A lo largo del proyecto se hace referencia a estas transformaciones por el nombre de transformaciones de color o transformaciones a nivel de píxel indistintamente.

[3]Suponiendo una profundidad de color de 8 bits.

**Transformaciones locales**

- **Desenfoque de caja (*box blur*):** cada píxel de la imagen resultante se calcula como el valor medio de sus píxeles vecinos en la imagen de entrada.
- **Desenfoque gaussiano (*gaussian blur*):** utiliza una función gaussiana (distribución normal) para calcular la transformación a aplicar a cada píxel de la imagen.

**Transformaciones globales**

- **Inyección de ruido:** transformación por la que a la imagen se le suma cierto *ruido* o valores diversos. Existen diversos tipos y por lo general buscan una distribución respecto la imagen global.
- **Ecualización del histograma:** transformación de la imagen de entrada en una imagen cuyo histograma [4] tenga una distribución uniforme.

## 2.2. Herramientas para *image data augmentation*

La tarea de *image data augmentation* no es algo nuevo, sino que existen múltiples herramientas, técnicas y librerías para llevarla a cabo. En este apartado hacemos un análisis de las principales herramientas existentes y sus características.

Para una comparación clara pero no exhaustiva, decidimos no analizar más de diez bibliotecas, y escoger aquellas que proporcionan características variadas para tener una visión completa de las funcionalidades disponibles. Las librerías que hemos analizado son:

- **Albumentations**: librería de código abierto enfocada a la tarea de *image data augmentation* y potenciación de redes neuronales convolucionales.
  https://github.com/albumentations-team/albumentations
- **ImgAug**: librería de código abierto centrada en el aumento de los *datasets* de imágenes con gran variedad de operaciones.
  https://github.com/aleju/imgaug
- **image_augmentor**: herramienta de código abierto orientada al aumento de *datasets* de imágenes en disco.
  https://github.com/codebox/image_augmentor

[4] El histograma de una imagen es un gráfico que muestra el número de píxeles de la imagen que hay con cada valor de intensidad posible. Por ejemplo, en una imagen con una profundidad de color de 8 bits (puede representar 256 intensidades diferentes) el histograma se compondrá de 256 columnas cuya altura represente respectivamente el número de píxeles con dicho valor.

- **TorchSample**: herramienta de Pytorch que ofrece operaciones para el entrenamiento de redes neuronales, para el *image data augmentation* y otras utilidades complementarias.
  https://github.com/ncullen93/torchsample

- **Scikit-image**: colección de algoritmos para el procesado de imágenes.
  https://scikit-image.org/

- **OpenCV**: biblioteca de acceso libre de visión por computador originalmente desarrollada por Intel.
  https://scikit-image.org/

- **TorchVision Transforms**: colección de *datasets* populares, arquitecturas y transformaciones de imágenes comunes para visión por computador.
  https://pytorch.org/docs/stable/torchvision/transforms.html

- **Kornia**: biblioteca de visión por computador diferenciable para PyTorch.
  https://github.com/kornia/kornia

- **NVIDIA DALI**: colección de módulos muy optimizados para las tarjetas gráficas NVIDIA, enfocada a acelerar el pre-procesamiento de los datos de entrada para aplicaciones de aprendizaje profundo.
  https://github.com/NVIDIA/DALI

Con el objetivo de realizar una comparación objetiva y general entre las herramientas anteriores hemos evaluado los siguientes aspectos:

➔ **Tipo de imagen**: tipo de dato o estructura Python con la que trabaja la herramienta para representar la imagen o dato de entrada. Por ejemplo *numpy array*, *tensor* o *pil image*.

➔ **Orientación al trabajo con redes neuronales**: si la herramienta está orientada a ser integrada con una herramienta de aprendizaje profundo (como *Keras* o *Pytorch*) o solo trabaja en la transformación de imágenes.

➔ **Soporte de generador de datos**: si la herramienta incluye alguna estructura para proporcionar las imágenes aumentadas a la red en lotes (*batches*) de varias imágenes a la vez.

➔ **Soporte metadatos**: si permite la transformación conjunta de imágenes y objetos bidimensionales asociados (coordenadas de puntos, mapas de calor, mapas de segmentación...)

➔ **Trabajo con GPU**: si la librería permite la realización de transformaciones sobre la GPU en lugar de sobre la CPU.

- ➔ **Transformaciones parametrizables**: si las transformaciones de la librería son totalmente parametrizables (se puede indicar grados de rotación, centro de rotación, probabilidad de aplicar la transformación...)
- ➔ **Uso de segundas librerías a bajo nivel**: si la herramienta emplea otras librerías de bajo nivel para las transformaciones.

Como se muestra en el Cuadro 2.1, existe gran diversidad en las propiedades de las librerías analizadas. El tipo de dato más empleado es `ndarray`, aunque también se emplea `PIL image` y en algún caso `torch tensor`. Muchas de las librerías se limitan a realizar las operaciones de *image data augmentation* (ImgAug, image_augmentor, scikit-image, opencCV...). No obstante, en nuestro caso es interesante que la propia herramienta integre soporte para trabajar directamente con librerías de *deep learning*. Respecto al uso de GPU, aun cuando las transformaciones geométricas se ven muy beneficiadas por una implementación sobre GPU debido a las multiplicaciones matriciales que implican, pocas librerías contemplan su utilización. Otra carencia notable es la realización del *image data augmentation* directamente sobre disco. Puede ser realmente interesante tener esta operación realizada de antemano (liberando la memoria de trabajo) para poder entrenar de forma eficiente la red (dedicando toda la capacidad de procesamiento al entrenamiento).

Cuadro 2.1: Comparativa de las características principales de librerías existentes para *image data augmentation*.

| | Tipo imagen | Orientado a DL | Soporte generador | Soporte metadatos | Soporte GPU | Generación a Disco | Parametrizable | Librerías |
|---|---|---|---|---|---|---|---|---|
| **Albumentations** | ndarray, dict | Pytorch | No | Si | No | No | Si | OpenCV |
| **ImgAug** | ndarray | No | No | Si | No | No | Si | Scikit + OpenCV |
| **Augmentor** | directorio (PIL interno) | Keras, Pytorch | Si | Máscaras | No (si multicore) | Si | Si | PIL |
| **image augmentor** | ndarray | No | No | No | No | Si (solo disco) | Si | Scikit |
| **Torch Sample** | torch tensor | Pytorch | Si | No | No | No | Si | No |
| **Scikit-image** | ndarray | No | No | Máscaras | No (simulticore | No | Si | No |
| **OpenCV** | ndarray | No | No | Si | Si | No | Si | No |
| **Torch Vision** | PIL Image | Pytorch | Si | Si | Si | Si | Si | No |
| **Kornia** | torch tensor | Pytorch | No | No | Si | No | Si | OpenCV |
| **NVIDIA DALI** | directorio, ndarray | Pytorch, Tensorflow, MXNet | Si | Si | Si + multicore | Si | Si | No |

## 2.3. Propuesta

En base al análisis del apartado anterior sobre características, ventajas y carencias de las opciones existentes en el mercado, la librería a desarrollar en este trabajo debe ofrecer un valor añadido al sector por los siguientes motivos:

- ✓ Se estudiará la composición de transformaciones,con el objetivo de obtener la máxima eficiencia posible.
- ✓ Se valorará y probará el rendimiento de la implementación de las operaciones sobre GPU optando por la opción más beneficiosa para el rendimiento de la librería.
- ✓ Se incluirá algún método de visualización que facilite la tarea de depurado de los programas que la integren (elemento que ninguna herramienta actual ofrece).
- ✓ Se permitirá la generación del *dataset* aumentado a disco y no únicamente sobre memoria, ya que puede interesar hacer por separado la tarea de *image data augmentation* y después emplear el *dataset* aumentado las veces que se quiera o de manera independiente.
- ✓ Se dará soporte a un *dataloader* o bien a un generador que permita su integración inmediata con librerías de *deep learning*. En concreto, se orientará a redes neuronales que trabajen con Pytorch ya que, de las librerías actuales de *deep learning*, es la que ofrece un mejor compromiso entre flexibilidad y facilidad de uso.
- ✓ Soportará transformaciones conjuntas de los datos bidimensionales más comunes (*keypoints, máscaras, mapas de segmentación y mapas de calor o saliencia)*
- ✓ Las operaciones serán totalmente parametrizables (orden de aplicación, probabilidad...).
- ✓ Será flexible, permitiendo cualquier combinación de tipos de datos, cualquier orden de operaciones...

Con esta combinación de características construiremos una librería completa que cubra todas las necesidades relacionadas con la tarea de *image data augmentation* de manera flexible, sencilla, y familiarizada con los términos y naturaleza del *deep learning*. De este modo se cubrirán las carencias detectadas en las herramientas existentes y se combinarán sus mayores ventajas en una única herramienta.

# Capítulo 3

# Gestión del proyecto

La gestión de proyectos está conformada por todas aquellas acciones necesarias para cumplir con un objetivo definido dentro de un período de tiempo determinado y unos recursos limitados. Es una tarea fundamental para el desarrollo del proyecto de una manera controlada, eficiente y de la manera más productiva posible.

En este capítulo se explican todos los aspectos relacionados con la gestión de este proyecto, incluyendo la metodología de desarrollo escogida, la gestión del alcance, de la configuración, de riesgos, costes y planificación temporal.

## 3.1. Metodología de desarrollo

La primera tarea a la hora de abordar un proyecto de software es elegir de la metodología de desarrollo que mejor se adapte a las necesidades del proyecto en cuestión. Una metodología de desarrollo adecuada nos permite entender cuál es nuestro rol dentro del proyecto y da un mayor control sobre sus diferentes posibilidades.

A la hora de escoger la metodología en este proyecto los aspectos más relevantes a tener en cuenta son:

1. Los requisitos y detalles de implementación del proyecto son muy susceptibles a cambios debido a su parte de investigación.

2. Dado que el objetivo del proyecto es lograr la máxima eficiencia, cobra gran importancia la revisión y *feedback* continuo sobre el trabajo realizado, ya que en base a los resultados obtenidos deben tomarse decisiones sobre el proyecto.

3. El proyecto va a incluir una parte importante de investigación, por lo que las decisiones de implementación se irán tomando a medida que se avance en el mismo. Por tanto, no es adecuado realizar en una etapa inicial la definición de su estructura ni detalles del mismo.

Las metodologías tradicionales quedan descartadas, ya que no ofrecer la flexibilidad que requiere el proyecto. Por este motivo hemos optado por una metodología ágil, ya que ofrece mayor flexibilidad y control frente a las incertidumbres inherentes al proyecto. También permite realizar cambios en la planificación, además de que se van asumiendo y revisando las necesidades cambiantes del proyecto en contacto continuo con el cliente. En concreto, la metodología ágil elegida es **Scrum** ya que es la más usada, documentada y conocida.

### 3.1.1. Scrum

Scrum se caracteriza por ser una metodología sencilla y flexible, y está basada en 3 pilares fundamentales:

1. **Transparencia**: dar visibilidad a todo lo que está pasando, ya que los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado.

2. **Inspección**: del progreso y resultados obtenidos para identificar y corregir las variaciones no deseadas.

3. **Adaptabilidad**: implica hacer los ajustes en los procesos y productos para minimizar la desviación.

Conocer estos pilares es más importante que seguir ciegamente la mecánica Scrum. Por lo general, debemos adaptar la metodología a las necesidades de cada proyecto, siguiendo unos principios y características comunes en todos los proyectos Scrum.

Las características generales en todo proyecto Scrum son:

- Se organizan los objetivos y el trabajo en ***sprints***. En Scrum no se aborda el proyecto como un global, sino que se van realizando iteraciones para ir consiguiendo pequeños productos de valor en tiempos cortos.

- Se da prioridad a lo que tiene más valor para el cliente.

- Cada vez que finaliza un *sprint*, debe llevarse a cabo una retrospectiva, analizando si el producto desarrollado satisface los criterios de aceptación fijados, si la organización fue adecuada y si el cliente da el visto bueno (o propone los cambios necesarios).

- Los equipos van a ser autoorganizados y autodirigidos, ellos mismos van a organizar sus tareas y se van a dirigir. Para ello, dentro del equipo se actúa según los roles asignados. Los **roles principales** en Scrum son:

    - **ScrumMaster**: mantiene los procesos y trabaja de forma similar al director de proyecto.

- **ProductOwner**: representa a los stakeholders (clientes externos o internos).
- **Team**: incluye a los desarrolladores.

Para llevar esto a la práctica, Scrum gira en torno a los siguientes elementos:

1. **Product Backlog**: lista de todas las funcionalidades o requisitos identificados para el producto. Organizados de manera priorizada en base a la importancia que suponen para el proyecto y para el cliente. Entre los elementos que conforman el *Product Backlog* están:
   - **Historias de usuario**: descripción breve y sencilla de una característica o funcionalidad deseada para el producto (contada desde la perspectiva de la persona que busca la nueva capacidad). Normalmente siguen el esquema `como _ quiero _ para _`. Por ejemplo, *`como` usuario de la librería `quiero` poder visualizar las transformaciones `para` depurar fácilmente mi código.*
   - **Épicas** o ***features***: historias de usuario que describen funcionalidades o características a alto nivel y que, al avanzar el proyecto, se van descomponiendo en un conjunto de historias de usuario más concretas.
   - **Spikes**: tareas que no implican desarrollar una historia de usuario y por tanto no aportan directamente un incremento. Es común la realización de tareas como:
     - ✓ Dedicar tiempo a qué tecnologías, librerías, *frameworks* se están utilizando para realizar proyectos como el que se aborda.
     - ✓ Hacer algunas pruebas con esas tecnologías para ver qué posibilidades ofrecen.
     - ✓ Formación teórica o investigación sobre técnicas asociadas a la naturaleza del proyecto.
2. ***Sprint backlog***: lista de tareas que se realizan en un *sprint*. Durante la planificación del *sprint* se eligen y asignan las tareas a los miembros del equipo.
3. **Incremento**: parte del producto que se desarrolla en un *sprint* (terminada y totalmente operativa). Representa los requisitos que se han completado en una iteración y que son operativos.

### Ciclo de vida

Como ya se ha mencionado, Scrum no aborda el proyecto como un global, sino que lo subdivide y afronta de manera iterativa en lo que denomina ***sprints***. Un

*sprint* es un intervalo prefijado de tiempo durante el cual se crea un incremento de producto y se trabaja para lograr unos subobjetivos concretos.

Antes de iniciar los *sprints*, se lleva a cabo una primera fase de preparación del proyecto. Conocida como *sprint* 0, es la fase en la que se intenta comprender el caso de negocio y objetivos generales del proyecto. Por lo general incluye tareas como:

- ✓ Definir el proyecto: indicar de forma clara el propósito del proyecto. No es necesario entrar en detalle pero sí que todo el equipo sea capaz de entender cuáles son las necesidades del producto y del cliente.

- ✓ Definir "terminado": marcará el punto en el que se va a considerar que la tarea está finalizada.

- ✓ Definición del *backlog* inicial: comenzar la creación del *backlog* del producto.

Una vez realizado el *sprint* 0, se inicia el desarrollo normal del proyecto, conformado como una sucesión de *sprints* consecutivos. A su vez, cada *sprint* suele subdividirse en 5 fases:

1. **Reunión de planificación de *sprint***: se define la funcionalidad en el incremento planeado y cómo el equipo de desarrollo creará este incremento.

2. **Scrum Diario**: evento de pocos minutos de duración, para que el equipo de desarrollo sincronice actividades y cree un plan para las próximas 24 horas.

3. **Trabajo de desarrollo durante el *sprint***: se trabaja en los objetivos fijados y siguiendo la planificación realizada.

4. **Revisión del *sprint***: se lleva a cabo al final del *sprint*, para inspeccionar el incremento y adaptar, si es necesario, el *product backlog*.

5. **Retrospectiva del *sprint***: momento para reflexionar sobre lo que ha ido bien, lo que podría hacerse mejor y lo que se podría perfeccionar en el siguiente *sprint*.


Figura 3.1: Ciclo de vida Scrum.

### 3.1.2. Aplicación de Scrum a este proyecto

Scrum debe adaptarse a las necesidades de cada proyecto y equipo en cuestión. En lo que refiere al proyecto que se describe en esta memoria, estas fueron las adaptaciones tomadas: en primer lugar, como el equipo de desarrollo se componía de una única persona, el Scrum diario no se ha llevado a cabo. Los *sprints* tienen una duración de 1 o 2 semanas según la dificultad de las tareas a abordar. Finalmente, respecto los roles, como el equipo de desarrollo está formado por una única persona, deberá asumir todas las tareas asociadas al *Team*, *ProductOwner* y *ScrumMaster*.

## 3.2. Gestión del alcance

En este apartado se detallan los aspectos relacionados con la gestión del alcance del proyecto, definiendo qué se incluye en el proyecto y qué no se incluye. Una gestión del alcance correcta permite centrarnos en aquellos aspectos esenciales para lograr el objetivo real del proyecto.

### 3.2.1. Descripción del alcance

La librería final debe satisfacer las necesidades de *image data augmentation* como complemento sencillo y flexible para programas de aprendizaje profundo y visión por computador. En concreto debe incluir las siguientes funcionalidades:

- Ofrecer una buena variedad de operaciones de transformación que satisfaga las necesidades del sector.
- Ofrecer un elemento de composición de estas transformaciones tipo *pipeline*. Que sus operaciones sean totalmente parametrizables (permitiendo fijar la probabilidad de aplicación de cada una). Este objeto debe ser capaz de transformar listas de datos de manera conjunta. A partir de este punto, se empleará el término *pipeline* para hacer referencia a este módulo de composición de operaciones.
- Ofrecer un objeto que integre el *pipeline* y sirva como suministro de elementos a una red neuronal artificial.
- Ofrecer la opción de generar el *dataset* aumentado directamente a disco, permitiendo así emplear el *dataset* aumentado de manera independiente a la librería.
- Ofrecer una herramienta de visualización de las transformaciones para facilitar la comprensión y depuración de programas.

### 3.2.2. Criterios aceptación

Es importante tener en cuenta que en Scrum, parte de la gestión del proyecto se lleva a cabo a lo largo de los *sprints* a medida que va avanzando el proyecto. En lo que refiere a los criterios de aceptación, a cada una de las historias de usuario se le asociarán sus criterios de aceptación específicos. En este apartado se detallan los criterios de aceptación del producto final.

Son condiciones necesarias para dar como aprobado el proyecto:

- **CA.01**: La herramienta permite la realización del proceso de *image data augmentation* sobre conjuntos de imágenes de cualquier tipo y tamaño sin generar errores (propios del código).
- **CA.02**: La herramienta ofrece un repertorio de transformaciones igual o superior al ofrecido por herramientas similares (10 o más operaciones).
- **CA.03**: La herramienta soporta transformaciones conjuntas sobre imágenes, mapas de calor o saliencia, máscaras, mapas de segmentación, coordenadas de puntos y cualquier combinación de los mismos de manera correcta y consistente.
- **CA.04**: La herramienta de visualización se despliega con éxito y presenta los datos de manera correcta.
- **CA.05**: La herramienta permite la realización del *image data augmentation* tanto sobre disco como sobre memoria volátil.

- **CA.06**: La herramienta puede considerarse una librería escalable, que respeta los estándares del lenguaje y las buenas prácticas asociadas, siguiendo la guía de estilo de Python **PEP 8** para asegurar un código limpio y legible en la medida de lo posible.

La herramienta será sometida a pruebas de rendimiento y funcionamiento que demuestren que se han alcanzado estos criterios de aceptación. Además, será examinada por los clientes en cada uno de los *sprints* y en su etapa final.

### 3.2.3. Restricciones del proyecto

En este apartado se detallan las limitaciones a las que se ve sometidas el proyecto:

- **RP.01**: La fecha de finalización límite del proyecto es el 6 de julio de 2020, para poder realizar su defensa en julio del mismo año.
- **RP.02**: La cantidad de trabajo realizado por el alumno no debe ser inferior a 400 horas ni superior a 425.

### 3.2.4. Exclusiones del proyecto

En este apartado se detallan los elementos que se consideran fuera del alcance del proyecto:

- **EP.01**: La herramienta se enfocará únicamente a su integración con la librería de aprendizaje profundo Pytorch por ser la que ofrece un mejor compromiso entre flexibilidad y facilidad de uso.
- **EP.02**: En ningún caso, la librería debe encargarse de ninguna tarea de *deep learning*, siendo solo un complemento para los programas del sector.

## 3.3. Gestión de la configuración

Esta sección detalla el plan de gestión de la configuración seguido durante el proyecto. Este plan se desarrolló para garantizar la calidad y la integridad de los productos desarrollados.

### 3.3.1. Elementos de configuración

En este apartado se detallan todos los componentes del proyecto identificados como elementos de configuración, y que, como tal, están sujetos a esta gestión:

- Código fuente de la librería.

- Manuales y documentación de la librería.
- Documentación asociada al proyecto software.
- Documentación de la investigación previa al proyecto.

### 3.3.2. Gestión de la configuración

Para el almacenamiento de los elementos de configuración (incluyendo la gestión de versiones y compartición con el cliente y equipo) se utilizan las siguientes plataformas:

- **Github**: plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git. Se crea un repositorio en Github para el desarrollo del código de manera supervisada y que hace el proyecto, en su fase final, público.
- **Read The Docs**: plataforma de alojamiento de documentación de software libre, en la que se hará pública y accesible la documentación de la librería en formato HTML, incorporando la gestión de versiones de la misma.
- **Overleaf**: editor online de Latex empleado para el almacenamiento, revisión y desarrollo de la memoria del proyecto.
- **OneDrive**: para los demás archivos y elementos de configuración del proyecto se crea un directorio compartido en la plataforma en la nube OneDrive, que permite compartir, revisar y gestionar versiones de los elementos incluidos.

## 3.4. Gestión del tiempo

En esta sección se detallan todos los procedimientos y planificación temporal llevados a cabo para la finalización en plazo del proyecto. Es importante tener en cuenta que en un proyecto Scrum no se hace una planificación temporal de todo el proyecto a nivel global. Tampoco se emplea tiempo en definir los paquetes de trabajo y requisitos de manera detallada antes de iniciar el proyecto como en las metodologías clásicas.

### 3.4.1. *Sprint Planning*

En Scrum se asume que el proyecto va a sufrir cambios. Por ello, la gestión del tiempo en Scrum se realiza de manera continua durante todo el desarrollo del proyecto. La planificación temporal se hace al inicio de cada *sprint*, siendo una

planificación de un período de tiempo y sobre un conjunto de requisitos mucho menor, haciendo así el desarrollo del proyecto resistente a cambios.

La **planificación del *sprint* o *sprint planning*** trata de responder a las siguientes preguntas:

- ¿Qué incremento podemos entregar en este *sprint*?
- ¿Cómo podemos entregar ese incremento?

En la primera parte, el equipo debe decidir qué historias de usuario y *spikes* del *Product Backlog* va a realizar en el siguiente *sprint*. En esta fase, es importante tener en cuenta las siguientes cuestiones:

- Es fundamental tener todos los ítems refinados, es decir, con el suficiente detalle como para que el equipo pueda entender, estimar y dividir cada ítem en tareas técnicas.
- Los *sprints* deben ser lo más cortos posibles, por ello para este proyecto los *sprints* no deben ser de más de dos semanas, siendo preferible 1 semana de duración.
- Cada ítem del *Product Backlog* debe tener una estimación temporal asignada y, de no tenerla, se debe estimar. Es importante no invertir demasiado tiempo en la estimación pero si tener en cuenta el rendimiento de los *sprints* anteriores para ser lo más realista posible.

Una vez seleccionadas, debe planificarse cómo se llevarán a cabo estas tareas. No es necesaria una planificación 100 % exacta pero sí lo suficiente para que cualquier miembro del equipo sepa lo que tiene que hacer en cada momento.

### 3.4.2. Fases del proyecto

Las historias de usuarios y *spikes* forman parte de la fase de desarrollo del proyecto. No obstante, es importante tener en cuenta otras fases que no se incluyen en los *sprints plannings*. Estas fases serán estimadas a priori para tenerlas en cuenta a lo largo de la evolución del proyecto:

- **Análisis y gestión del proyecto** - Estimación: 1,5 semanas de duración. Incluye las tareas iniciales para la definición de las características deseadas en el proyecto, así como las tareas de gestión del proyecto. Debe hacerse antes del inicio del desarrollo.
- **Documentación del proyecto**: - Estimación: 3 semanas de duración. Incluye las tareas de documentación propias del TFG como son la redacción del anteproyecto, de la memoria y de la presentación pública.

Debido a la situación excepcional del estado de alarma sanitario por el COVID-19, la planificación se modificó para realizar el proyecto en una duración de 14 semanas, dedicando 29 horas de trabajo semanales.

Como la tarea de documentación consume mucho tiempo y es vital que se fuese realizando progresivamente a lo largo del desarrollo del proyecto,en cada uno de los *sprints* se añaden 0,5 semanas de trabajo adicional dedicados en exclusiva a la documentación.

### 3.4.3. Estructura de descomposición del trabajo final (EDT)

En el EDT de la Figura 3.2 se incluyen las actividades de estas dos fases y representan la división de trabajo final que se asoció a cada *sprint*. Los detalles de desarrollo, diseño, análisis e implementación de cada *sprint* se encuentran en el capítulo 5 de esta memoria. A continuación se detalla brevemente la información de evolución del proyecto y duración de cada fase:

- ***Sprint* 0**: 0,5 semanas de duración.
  Se definieron los objetivos a alto nivel del proyecto. En concreto, tuvo lugar la reunión inicial con los tutores y se definió la estructura inicial del *product backlog* a partir de las épicas identificadas. También tuvo lugar la tarea de formación previa en las tecnologías de *deep learning*, para entender mejor las necesidades y funcionamiento del sector.

- **Gestión del proyecto**: 1,5 semanas de duración.
  Una vez definidos a alto nivel los objetivos del proyecto, se llevó a cabo la tarea de gestión del proyecto para planificar y gestionar de manera adecuada todos los aspectos relevantes del mismo.

- ***Sprint* 1**: 2 semanas de duración (+ 0,5 semanas de documentación).
  Este *sprint* se dedicó fundamentalmente a investigación y formación. En primer lugar se estudiaron las técnicas y posibles implementaciones para las transformaciones tanto geométricas como de color. A continuación se realizó un estudio de las principales características de herramientas del sector y un estudio de su rendimiento para seleccionar las más adecuadas para el proyecto.

- ***Sprint* 2**: 1 semana de duración (+ 0,5 semanas de documentación).
  Se aplicaron las conclusiones extraídas de la formación e investigación previa para implementar las operaciones individuales, tanto geométricas como de color.

- ***Sprint* 3**: 2 semanas de duración (+ 0,5 semanas de documentación).
  Se construyó una primera versión del *pipeline* que se encargase de la composición de las operaciones. Paralelamente se fue construyendo la herramienta de visualización para las operaciones independientes.

- ***Sprint* 4**: 1 semana de duración (+ 0,5 semanas de documentación).
  Una vez construido el *pipeline* inicial, se trabajó en hacerlo más flexible, incluyendo parámetros de personalización (modo de interpolación, de extrapolación,formato..). Paralelamente, sobre el *pipeline* ya construido se añadió la función de visualización (con la transformación de lotes de imágenes que incluye el *pipeline*) basándose en el trabajo ya realizado para la visualización individual.

- ***Sprint* 5:** 2 semanas de duración (+ 0,5 semanas de documentación).
  Este *sprint* se inició con la tarea de análisis de los objetos *dataloader* y *generadores*. A continuación se implementó el objeto *dataloader* propio de la librería, que incorporaba nuestro *pipeline* en su interior. De manera análoga y aprovechando la formación obtenida en estos objetos se diseñó e implementó también el módulo de *image data augmentation* sobre disco que incorporaba también el *pipeline* en su interior.

- **Sprint 6:** 1 semana de duración (+ 0,5 semanas de documentación).
  En este *sprint*, se centraron los esfuerzos en la finalización de la API. En primer lugar se llevó a cabo una reestructuración de la librería de forma que cumpliese con los estándares establecidos por la organización PyPA en la guía (*Python Packaging User Guide*). A nivel de código se llevó también a cabo una *refactorización* del código para que cumpliese las buenas prácticas de la guía **PEP 8** de Python. Una vez limpio y organizado, se generó la documentación propia de la librería empleando la herramienta Sphinx. Finalmente, se hace públicamente accesible e instalable la API mediante la herramienta `PyPI` y la documentación pública en el repositorio ReadTheDocs.

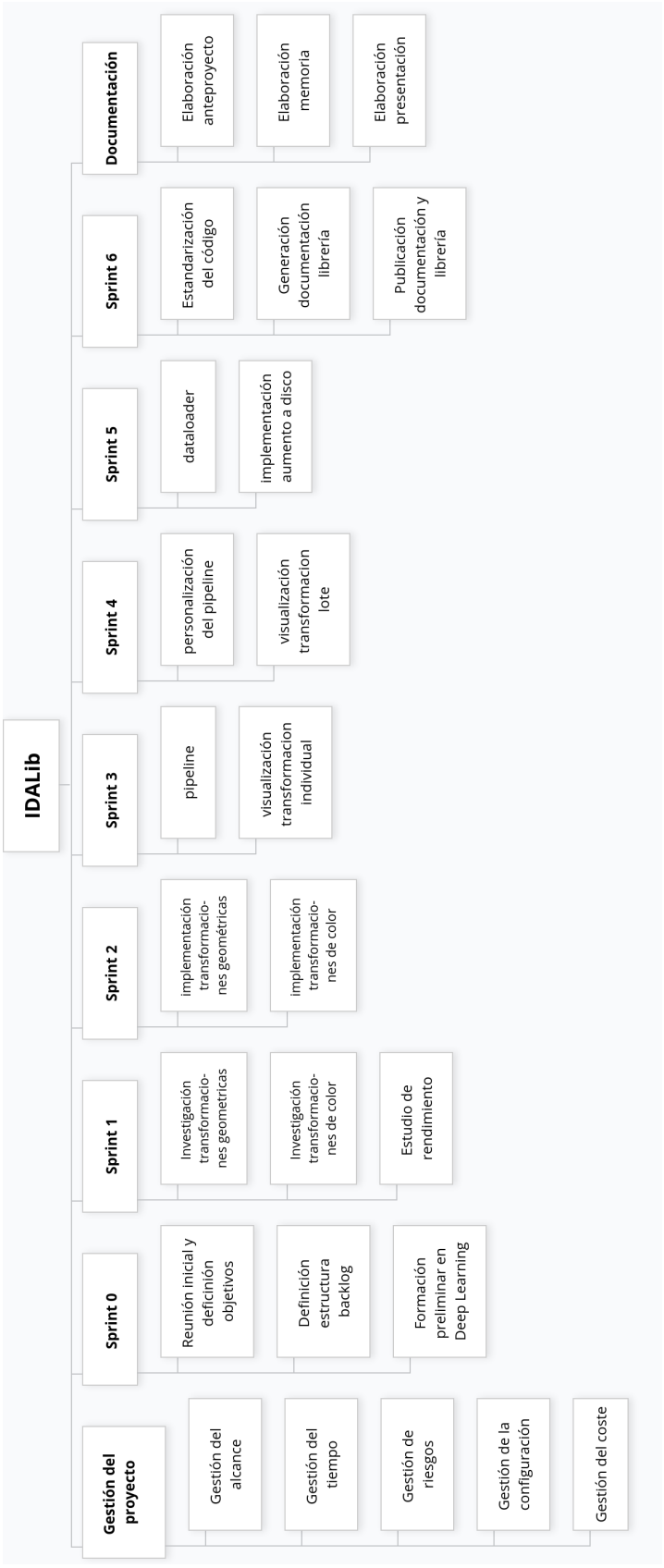



Figura 3.2: EDT del proyecto IDALib.

## 3.5. Gestión de riesgos

Como cualquier proyecto de software, éste está expuesto a riesgos que ponen en peligro el correcto desarrollo y finalización del mismo. En este apartado se detallan los procesos que se llevan a cabo para identificar, analizar y responder a los riesgos que se expone el proyecto.

### 3.5.1. Metodología de análisis de riesgos

Para un análisis consistente de los riesgos a los que está expuesto el proyecto se lleva a cabo una **priorización cualitativa de riesgos**, es decir, a cada uno de los riesgos se le asigna un valor de probabilidad siguiendo el Cuadro 3.1 y un valor de impacto (Cuadro 3.2). Con estos dos valores se calcula la exposición del mismo (Cuadro 3.3), que sirve como parámetro de priorización.

Cuadro 3.1: Valoración de la probabilidad de riesgo.

| **Probabilidad** | |
|---|---|
| Alta | Debe asumirse que se va a producir en algún momento del desarrollo del proyecto, incluso en más de una ocasión. |
| Media | Deberá asumirse que alguno de los riesgos con esta probabilidad se manifieste durante el desarrollo del proyecto. |
| Baja | Solo se dará en situaciones excepcionales. Podemos asumir que no se va a dar durante el desarrollo del proyecto. |

Cuadro 3.2: Valoración del impacto de riesgo.

| **Impacto** | |
|---|---|
| Catastrófico | No se puede seguir adelante con el TFG. |
| Serio | Supone retrasos considerables en la entrega, pudiendo suponer el atrasar la entrega una convocatoria. |
| Tolerable. | Se puede resolver sin grandes impactos en la planificación. |

Cuadro 3.3: Cálculo de la exposición de riesgo.

| **Exposición** | | **Probabilidad** | | |
|---|---|---|---|---|
| | | Alta | Media | Baja |
| **Impacto** | Catastrófico | Alta | Media | Baja |
| | Serio | Media | Media | Baja |
| | Tolerable | Baja | Baja | Baja |

### 3.5.2. Identificación de riesgos

En esta sección se detallan los riesgos identificados a los que se ve expuesto el proyecto. Cada uno de los riesgos se va a describir y analizar en base a su impacto y probabilidad en el proyecto.

Para la identificación y análisis de los riesgos se va a emplear el formato del Cuadro 3.4:

Cuadro 3.4: Documento guía. Análisis de riesgo.

| **Código**- Nombre del Riesgo | |
|---|---|
| **Descripción** | Breve descripción del riesgo. |
| **Probabilidad** | Alta, media o baja. |
| **Impacto** | Tolerable, serio, catastrófico. |

| **RSG.01**- La curva de aprendizaje de las tecnologías a emplear es más larga de lo esperado. | |
|---|---|
| **Descripción** | La estimación de tiempo que requiere la formación en las tecnologías y técnicas usadas para el proyecto es demasiado optimista y en la práctica requiere más tiempo, suponiendo retrasos en la planificación. |
| **Probabilidad** | Media. |
| **Impacto** | Serio. |

| **RSG.02**- La herramienta no satisface las necesidades asociadas al sector del aprendizaje profundo que requiera tareas de *image data augmentation*. | |
|---|---|
| **Descripción** | La herramienta desarrollada no cumple con las necesidades del sector y su uso final no resulta práctico. |
| **Probabilidad** | Media. |
| **Impacto** | Serio. |

| **RSG.03**- El desarrollo de funciones software no es aprobado por el cliente y requiere volver a diseñarlas e implementarlas. | |
|---|---|
| **Descripción** | El cliente no da el visto bueno a la implementación de determinadas funcionalidades, requiriendo rehacerlas, lo que conlleva a retrasos en la planificación. |
| **Probabilidad** | Alta. |
| **Impacto** | Serio. |

| **RSG.04**- Planificación optimista (en lugar de realista). | |
|---|---|
| **Descripción** | Se subestima el tiempo que requieren las tareas en la planificación del proyecto, siendo inviable su realización según lo planificado. |
| **Probabilidad** | Alta. |
| **Impacto** | Catastrófico. |

| **RSG.05**- Las tareas de investigación no son concluyentes. | |
|---|---|
| **Descripción** | Las tareas de investigación necesarias para las decisiones de diseño no son concluyentes. |
| **Probabilidad** | Alta. |
| **Impacto** | Tolerable. |

| **RSG.06**- Pérdida de datos del proyecto. | |
|---|---|
| **Descripción** | Por error humano o tecnológico se provoca la pérdida de datos del proyecto, ya sean de documentación o de código fuente. |
| **Probabilidad** | Alta. |
| **Impacto** | Serio. |

| **RSG.07**- El desarrollo de funciones software innecesarias alarga la planificación. | |
|---|---|
| **Descripción** | Se dedica tiempo de desarrollo a funcionalidades que no están dentro del alcance del proyecto causando retrasos en elementos que si lo están. |
| **Probabilidad** | Media. |
| **Impacto** | Serio. |

| **RSG.08**- EL cliente no acepta el software entregado, incluso cumpliendo todas sus especificaciones. | |
|---|---|
| **Descripción** | El cliente no está satisfecho con el resultado final del producto y no lo acepta. |
| **Probabilidad** | Baja. |
| **Impacto** | Catastrófico. |

| **RSG.09**- Las pruebas diseñadas no se pueden llevar a cabo en el equipo disponible. | |
|---|---|
| **Descripción** | Las pruebas de rendimiento diseñadas suponen una capacidad de procesamiento o de memoria de la que no se dispone. |
| **Probabilidad** | Alta. |
| **Impacto** | Serio. |

| **RSG.10**- Las partes del proyecto que no se han especificado claramente consumen más tiempo del esperado. | |
|---|---|
| **Descripción** | La planificación de partes del proyecto poco definidas no se ajusta a la realidad, causando retrasos en la planificación. |
| **Probabilidad** | Alta. |
| **Impacto** | Serio. |

| **RSG.11**- Se detectan errores de diseño en una fase avanzada del proyecto. | |
|---|---|
| **Descripción** | En fases avanzadas del desarrollo del proyecto se detectan problemas causados por errores de diseño en las fases iniciales. Esto causa un rediseño de todo el trabajo posterior y la reimplementación de elementos necesarios (y su correspondiente retraso). |
| **Probabilidad** | Media. |
| **Impacto** | Serio. |

| **RSG.12**- Detección de fallos importantes en los resultados tras la validación. | |
|---|---|
| **Descripción** | Se detectan errores importantes en la fase final de validación que no pueden ser corregidos en plazo. |
| **Probabilidad** | Media. |
| **Impacto** | Catastrófico. |

| **RSG.13**- Falla la integración de la herramienta desarrollada con los *frameworks* de aprendizaje profundo. | |
|---|---|
| **Descripción** | La integración de la herramienta desarrollada con *frameworks* de *deep learning* falla por incompatibilidades de formato u otros motivos. |
| **Probabilidad** | Alta. |
| **Impacto** | Serio. |

| **RSG.14**- La herramienta desarrollada no ofrece ninguna ventaja frente las herramientas existentes. | |
|---|---|
| **Descripción** | La herramienta final no ofrece ninguna ventaja frente otras herramientas disponibles y no hay motivos para su uso. |
| **Probabilidad** | Media. |
| **Impacto** | Serio. |

| **RSG.15**- Computador de desarrollo estropeado. | |
|---|---|
| **Descripción** | El equipo empleado para el desarrollo del proyecto sufre una avería y no se puede cumplir la planificación. |
| **Probabilidad** | Baja. |
| **Impacto** | Catastrófico. |

| **RSG.16**- La publicación de la librería en el repositorio PyPI falla. | |
|---|---|
| **Descripción** | Las condiciones de formato y referencias para poder publicar la librería en el repositorio PyPI (para poder ser instalable en cualquier equipo) no se cumplen siendo imposible su publicación. |
| **Probabilidad** | Media. |
| **Impacto** | Tolerable. |

| **RSG.17**- No se puede construir un producto de tal envergadura en el tiempo asignado. | |
|---|---|
| **Descripción** | El alcance del proyecto planteado es demasiado grande para los recursos y tiempo disponibles. |
| **Probabilidad** | Media. |
| **Impacto** | Catastrófico. |

| **RSG.18**- En el desarrollo, las nuevas funcionalidades hacen fallar las ya implementadas. | |
|---|---|
| **Descripción** | Las modificaciones llevadas a cabo para incluir las nuevas funcionalidades en la librería provocan errores en las funcionalidades previas. |
| **Probabilidad** | Alta. |
| **Impacto** | Serio. |

| **RSG.19**- Alguna de las plataformas *online* empleadas no está disponible. | |
|---|---|
| **Descripción** | Alguna de las plataformas *onlines* empleadas para el desarrollo del proyecto sufre una caída temporal y no están disponibles para su uso. |
| **Probabilidad** | Alta. |
| **Impacto** | Tolerable. |

| **RSG.20**- La metodología de desarrollo seleccionada no se adapta bien al proyecto. | |
|---|---|
| **Descripción** | La metodología seleccionada no se adapta a las necesidades del proyecto en cuestión dificultando la realización del mismo. |
| **Probabilidad** | Media. |
| **Impacto** | Tolerable. |

### 3.5.3. Planificación de respuesta a riesgos

#### Matriz de exposición

Siguiendo el formato del Cuadro 3.3 se clasifican los riesgos analizados según su exposición como se puede ver en el Cuadro 3.5.

Cuadro 3.5: Matriz de exposición de riesgos.

| Imp\Prob | **Alta** | **Media** | **Baja** |
|---|---|---|---|
| **Catastrófico** | RSG.04 | RSG.12<br>RSG.17 | RSG.08<br>RSG.15 |
| **Serio** | RSG.03<br>RSG.06<br>RSG.09<br>RSG.10<br>RSG.13<br>RSG.18 | RSG.01<br>RSG.02<br>RSG.07<br>RSG.11<br>RSG.14 | |
| **Tolerable** | RSG.05<br>RSG.19 | RSG.16<br>RSG.20 | |

Esta matriz de exposición ayuda a priorizar los riesgos que debemos tratar. En este caso, como solo existe un riesgo de exposición alta, podemos destinar recursos al tratamiento de riesgos de exposición media. En este proyecto vamos a tratar los siguientes riesgos:

1. **RSG.04**: planificación optimista (en lugar de realista).

2. **RSG.03**: el desarrollo de funciones software no es aprobado por el cliente y requiere volver a diseñarlas e implementarlas.

3. **RSG.06**: pérdida de datos del proyecto.

4. **RSG.09**: las pruebas diseñadas no se pueden llevar a cabo en el equipo disponible.

5. **RSG.10**: las partes del proyecto que no se han especificado claramente consumen más tiempo del esperado.

6. **RSG.12**: detección de fallos importantes en los resultados tras la validación.

7. **RSG.13**: falla la integración de la herramienta desarrollada con los *frameworks* de *deep learning*.

8. **RSG.17**: no se puede construir un producto de tal envergadura en el tiempo asignado.

9. **RSG.18**: en el desarrollo, las nuevas funcionalidades hacen fallar las ya implementadas.

### Acciones planificadas

Para cada riesgo, se han planificado una o más acciones correctivas y/o preventivas, y se han definido una serie de indicadores para permitir la detección temprana de su materialización. A continuación se muestra, para cada riesgo a tratar, las acciones planificadas:

| **RSG.04** | **Planificación optimista (en lugar de realista).** |
|---|---|
| Indicador | Se producen retrasos del más del 25 % del tiempo respecto la planificación original. |
| Prevención | Tratar de hacer una planificación en el peor de los casos. |
| Corrección | Reducir el alcance del proyecto y replanificar las tareas restantes. |

| **RSG.03** | **El desarrollo de funciones software no es aprobado por el cliente y requiere volver a diseñarlas e implementarlas.** |
|---|---|
| Indicador | El cliente rechaza elementos del producto desarrollado. |
| Prevención | Seguir la metodología Scrum y hacer al cliente partícipe de todos los avances, pudiendo aceptar los elementos desarrollados o proponer los cambios necesarios para hacer el producto a medida de sus intereses. |

| **RSG.06** | **Pérdida de datos del proyecto.** |
|---|---|
| Indicador | Parte o el total de los documentos se pierde por errores del equipo o de los repositorios de la nube. |
| Prevención | Todos los elementos de configuración del proyecto así como elementos necesarios serán almacenados tanto en la nube (ya sea OneDrive para archivos como GitHub para código) y sincronizados con una versión idéntica en el computador de desarrollo. |

| **RSG.09** | **Las pruebas diseñadas no se pueden llevar a cabo en el equipo disponible.** |
|---|---|
| Indicador | El equipo se bloquea o es incapaz de finalizar la computación de las pruebas diseñadas. |
| Prevención | Las pruebas se diseñarán teniendo en cuenta las limitaciones del equipo de desarrollo, nunca superando el 85 % de su capacidad. |
| Corrección | Se podrá hacer uso de un equipo secundario que cuenta con una capacidad de memoria y procesamiento superior. |

| **RSG.10** | **Las partes del proyecto que no se han especificado claramente consumen más tiempo del esperado.** |
|---|---|
| Indicador | Alguna de las tareas del *backlog* consume más del 25 % del tiempo previsto. |
| Prevención | Emplear metodología *Scrum* permite ir definiendo con más detalle los elementos poco definidos en fases más avanzadas del proyecto, pudiendo adaptar el alcance a los requerimientos de los requisitos más importantes para el cliente. |
| Corrección | Reformular el alcance del proyecto para lograr los objetivos más importantes para el cliente. |

| **RSG.12** | **Detección de fallos importantes en los resultados tras la validación.** |
|---|---|
| Indicador | Durante la validación se detectan errores que hacen que el producto no cumpla con los criterios de aceptación. |
| Prevención | Emplear metodología *Scrum* e ir validando y *testeando* el correcto funcionamiento y aceptación del cliente de cada parte construida, haciendo una validación progresiva a lo largo de todo el desarrollo. |

| **RSG.13** | **Falla la integración de la herramienta desarrollada con los *frameworks* de *deep learning*.** |
|---|---|
| Indicador | No es posible la construcción de un programa funcional de *deep learning* que integre la librería desarrollada. |
| Prevención | Dedicar un 25 % más de tiempo a la tarea de formación en el funcionamiento y necesidades de los *frameworks* de *deep learning* como Pytorch. |

| **RSG.17** | **No se puede construir un producto de tal envergadura en el tiempo asignado.** |
|---|---|
| Indicador | Después de 2 sprints analizar el rendimiento demostrado permite determinar que las tareas que aun quedan pendientes no se pueden completar en el periodo restante. |
| Prevención | Investigación previa sobre el sector para conocer la envergadura del proyecto a priori. Consultar a expertos del campo. |
| Corrección | Reformular el alcance del proyecto para limitar la envergadura del proyecto a un tamaño realizable y dejar lo demás como posible continuación. |

| **RSG.18** | **En el desarrollo, las nuevas funcionalidades hacen fallar a las ya implementadas.** |
|---|---|
| Indicador | La ejecución de funcionalidades ya implementadas presentan errores o comportamientos poco regulares. |
| Prevención | Aplicar metodología *Scrum* e incorporar pruebas asociadas a cada funcionalidad. Al incorporar nuevas funcionalidades debe comprobarse que se superan estas pruebas sin error y, de haber error, detectarlo y corregirlo lo antes posible. |

### 3.5.4. Riesgos materializados

Por lo general, el desarrollo del proyecto se llevó a cabo sin inconvenientes y de manera consistente a lo planificado. En cierta medida si se materializó el **RSG.18** ya que, siendo módulos dependientes, al agregar nuevas funcionalidades era común insertar errores en las anteriores. No obstante, gracias al plan de prevención estos errores eran detectados al momento mediante la ejecución de las pruebas unitarias y su corrección fue siempre sencilla sin afectar a la planificación.

También el **RSG.06** tuvo lugar debido a que errores en el equipo de desarrollo causaron la pérdida de determinados documentos del proyecto en local, lo cual no fue un inconveniente gracias al plan de prevención por el que se disponía de una copia actualizada en OneDrive.

Por otro lado, también tuvieron lugar algunos riesgos que no se habían identificados como posibles debido a la declaración de la crisis sanitaria del COVID-19: no fue posible llevar a cabo las reuniones tal y como se había planificado; no obstante, se respondió de manera sencilla haciendo reuniones *online*.

## 3.6. Gestión de costes

En este apartado se detalla la gestión de los costes asociados al proyecto. Esta es una tarea esencial para asegurar la viabilidad económica del proyecto y la gestión eficiente de sus recursos.

Para un análisis global de los costes del proyecto se tendrán en cuenta las siguientes fuentes de coste:

- **Costes de recursos humanos**: importe destinado a pagar a los miembros del equipo.
- **Costes de material**: importe destinado a la adquisición de equipos, infraestructuras, licencias y cualquier elemento material externo necesario para la realización del proyecto.
- **Costes indirectos**: costes necesarios para la realización del proyecto pero que no pueden ser atribuibles a un único proyecto y por lo tanto, no se pueden estimar de forma exacta.

### 3.6.1. Costes de recursos humanos

En primer lugar se analizan los costes relacionados al equipo de trabajo del proyecto. En este caso, el equipo está formado únicamente por la alumna, que se entiende que asume el rol de analista-programadora.

El salario de este miembro asciende a **16.000 € brutos anuales** (basándonos en la cifra de la Guía Salarial del Sector TIC en Galicia 2019-2020 [18]. No obstante, el salario mínimo fijado en la Normativa de la Universidade de Santiago de Compostela para la contratación de personal con cargo a actividades de investigación, es de 17.820,88€ por la jornada completa (40 horas semanales).

Este sueldo debe adaptarse a las 29 horas semanales que llevará a cabo la alumna, quedando un salario estimado de 13.365,66€ brutos anuales, lo que es equivalente a un salario bruto de 1.114€ mensual con pagas extras prorrateadas (en 12 pagas).

Para el cálculo se emplea la calculadora online de contratos de la Oficina de Investigación y Tecnología de la USC [1] :

- Número de pagas: 12

[1] http://imaisd.usc.es/ferramentas/calculadora/calculadoracontratos.asp

- Bruto mensual: 1.114€
- Horas semanales: 29
- Categoría: Grado - Investigador en formación
- **Coste total de 5.847€** de los cuales 1.402,74 son la aportación a la seguridad social.

No se tienen en cuenta como costes el trabajo realizado por parte de los tutores ya que, en este proyecto, asumen el rol de clientes.

### 3.6.2. Costes de material

Para la realización de este proyecto no fueron necesarios grandes requerimientos materiales, únicamente se empleó el equipo portátil de la alumna.

Suponiendo que la vida útil de un ordenador es de 4 años, y el portátil empleado tiene un coste de 1.095€, con un valor residual aproximado de 50,00€ y un coeficiente de amortización lineal del 33 %, el coste de amortización viene dado por:

$$(precio\ de\ compra - Valor\ residual) \times Coeficiente\ de\ amortización$$
$$(1095,00€ - 50,00€) \times 0,333 = 347,98€$$

Ponderando este coste por el número de horas dedicadas al proyecto con respecto al número de horas anuales (52 semanas) de uso del portátil (35 horas semanales), el coste del material atribuido al proyecto es:

$$347,98€ \times (401/2080) = 67,09€ \tag{3.1}$$

### 3.6.3. Costes indirectos

Como ya se mencionó, los costes indirectos son costes necesarios para la realización del proyecto pero que no se pueden atribuir explícitamente al mismo. Por ejemplo son costes indirectos el gasto en electricidad, internet, calefacción... Para estimar este coste se empleará la cifra que establece la USC para proyectos TIC, indicado en un 21 % adicional sobre los costes directos del proyecto.

### 3.6.4. Coste total del proyecto

Teniendo en cuenta todas las fuentes de costes del proyecto, podemos estimar el coste total desplegado en el Cuadro 3.6:

Cuadro 3.6: Costes totales del proyecto.

| **Recurso** | **Coste total** |
|---|---|
| Recursos humanos | 5.847€ |
| Recursos materiales | 67,09€ |
| Costes indirectos (21 %) | 1.241,95€ |
| **Total** | **7.156,04€** |

# Capítulo 4

# Análisis

En los ciclos de vida tradicionales, la fase de análisis tiene como objetivo especificar los requisitos funcionales, no funcionales, de información y cualquier requerimiento del proyecto de forma precisa y detallada para entender y definir qué es lo que busca el cliente.

En el caso de Scrum, no se invierte tanto tiempo en una descripción detallada, sino que se opta por dejar claros cuáles son los objetivos a alto nivel que el cliente espera del producto y, a medida que avanza el proyecto, estos se van refinando según la visión del cliente. De esta manera, se ahorra tiempo dedicado a la especificación de detalles y se implica al cliente en la evolución del proyecto.

En este apartado se va a detallar el *Product Backlog*, que representa las funcionalidades esperadas por el cliente para el producto. Además, se detallará el análisis de herramientas empleadas.

## 4.1. *Product Backlog*

En Scrum se definen una serie de historias de usuario (como se explicó en el apartado 3.1.1) que especifican las funcionalidades que espera el cliente del sistema. Estas historias de usuario se organizan en el ***product backlog*** de manera priorizada para tener una visión clara de las tareas finalizadas y pendientes de realización.

### 4.1.1. Épicas o *Features*

En una primera fase, las historias de usuario serán muy genéricas, dado que representan características de la aplicación a alto nivel. A medida que se avanza en el proyecto y se posee un conocimiento mayor del mismo, se van dividiendo en historias de usuario más espedíficas.

En Scrum se entiende como épica o *feature* un módulo o conjunto de historias de usuario que trabajan por una finalidad común.

Las épicas definidas para este proyecto son:

| **FTR.01** - Operaciones de transformación | |
|---|---|
| **Como** | usuario de IDALib |
| **Quiero** | poder hacer transformaciones de manera directa sobre imágenes y objetos bidimensionales |
| **Para** | usar en cualquier programa de propósito general. |

| **FTR.02** - Visualizar transformaciones | |
|---|---|
| **Como** | usuario de IDALib |
| **Quiero** | poder visualizar de manera sencilla las transformaciones que aplico |
| **Para** | depurar fácilmente mis programas. |

| **FTR.03** - *Pipeline* | |
|---|---|
| **Como** | usuario de IDALib |
| **Quiero** | poder componer transformaciones a las que pueda fijar una probabilidad y pasar lotes de imágenes por ella |
| **Para** | realizar el *image data augmentation* de manera sencilla. |

| **FTR.04** - *Dataloader* | |
|---|---|
| **Como** | usuario IDALib |
| **Quiero** | poder usar directamente un objeto que realice el *image data augmentation* sobre mi *dataset* y sirva como entrada a mi red neuronal a modo de generador |
| **Para** | incorporar el *image data augmentation* de forma sencilla a mi red neuronal. |

| **FTR.05** - *Image data augmentation* a disco | |
|---|---|
| **Como** | usuario IDALib |
| **Quiero** | generar un *dataset* aumentado a disco |
| **Para** | poder utilizarlo de manera independiente. |

| **FTR.06** - Publicación y estandarización | |
|---|---|
| **Como** | usuario IDALib |
| **Quiero** | poder instalar IDALib y disponer de un manual de usuario |
| **Para** | que su utilización sea sencilla. |

### 4.1.2. Estructura del *product backlog*

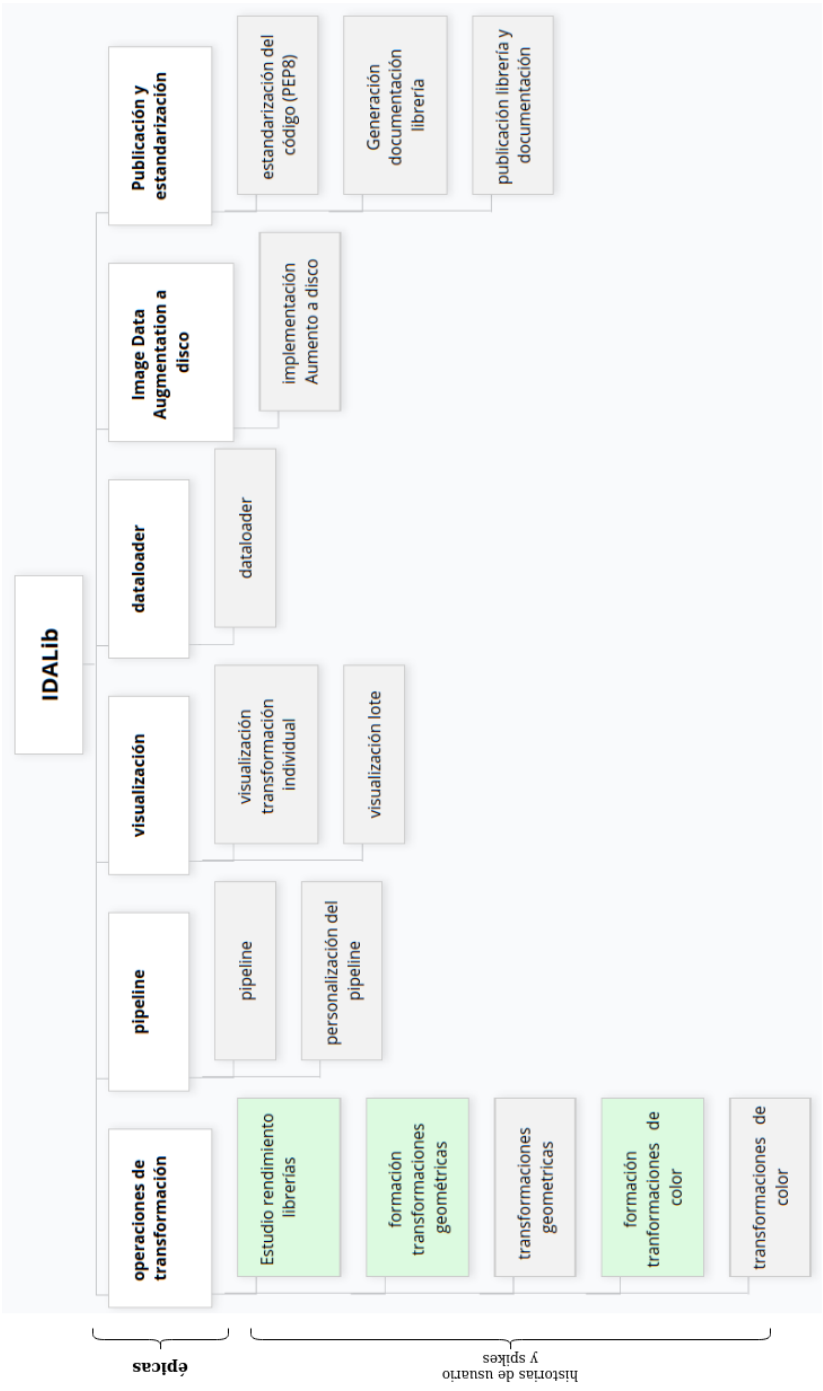



Figura 4.1: Estructura del *product backlog*, donde las historias de usuario en se representan en gris y los *spikes* en verde.

La estructura del *product backlog* del proyecto se muestra en la Figura 4.1. Los elementos de la primera fila se corresponden con las épicas ya explicadas. A cada una de ellas se le asocian las historias de usuario (en gris) y *spikes* (en verde) en los que se fue dividiendo cada épica durante el desarrollo del proyecto.

### 4.1.3. *Spikes*

Los *spikes* son los elementos empleados para reflejar trabajos necesarios en el proyecto pero que no constituyen un incremento del producto. Los *spikes* implicados en este proyecto son:

| **SPK.01** - Formación en transformaciones espaciales (geométricas) | |
|---|---|
| **Objetivo** | Obtener la formación suficiente sobre las diferentes implementaciones de transformaciones geométricas y herramientas existentes para conseguir una implementación correcta y eficiente. |

| **SPK.02** - Formación en transformaciones a nivel de píxel (de color) | |
|---|---|
| **Objetivo** | Obtener la formación suficiente sobre las diferentes implementaciones de transformaciones de color y herramientas existentes para conseguir una implementación correcta y eficiente. |

| **SPK.03** - Estudio de rendimiento de librerías | |
|---|---|
| **Objetivo** | Llevar a cabo la investigación y pruebas de rendimiento necesarias para determinar las librerías de bajo nivel que mejor se adapten a las necesidades de implementación de las transformaciones a bajo nivel, tanto espaciales como de color. |

### 4.1.4. Historias de Usuario

Para definir de forma correcta las historias de usuario a partir de las épicas es necesario que cumplan las siguientes condiciones:

- Deben ser realizables en un *sprint*, es decir, su duración estimada no debe ser nunca mayor a 2 semanas (para este proyecto).
- Deben ser lo suficiente precisas como para que el equipo pueda entender, estimar y dividir cada ítem en tareas concretas.
- Deben aportar valor al proyecto y al cliente.
- Deben ser verificables.

Para especificar las historias de usuario se les asignará un código, un nombre, la descripción en formato (como-quiero-para), la lista de tareas en las que se descompone, el valor de prioridad que tienen para el proyecto y el criterio de aceptación de la misma. Respecto al nivel de prioridad, distinguimos tres niveles:

- **Alta** (Cumplimiento obligatorio): es imprescindible que esté en la versión final para poder aceptar el proyecto.
- **Media** (Cumplimiento deseable): es deseable en la versión final del producto, pero no supone una funcionalidad esencial del mismo.
- **Baja** (Cumplimiento opcional): su presencia aporta cierto valor al cliente pero no es crucial para el éxito del proyecto. Es posible posponerla como ampliación futura.

A continuación se detallan las historias de usuario definidas en este proyecto, siguiendo la estructura del Cuadro 4.1:

Cuadro 4.1: Estructura de la especificación de las historias de usuario.

| **Código de historia** - Nombre de la historia de usuario | | |
|---|---|---|
| **Descripción** | **Como** | tipo de usuario que solicita la historia |
| | **Quiero** | funcionalidad deseada |
| | **Para** | finalidad en que emplea la funcionalidad. |
| **Prioridad** | | Alta, media o baja. |
| **Duración** | | Duración estimada en semanas. |
| **Aceptación** | | Criterio de aceptación de la historia de usuario. |
| **Tareas** | | Lista de tareas en que se descompone la historia. |

| **HU.01** -Transformaciones geométricas | | |
|---|---|---|
| **Descripción** | **Como** | usuario IDALib |
| | **Quiero** | poder hacer transformaciones geométricas de manera directa sobre imágenes y objetos bidimensionales |
| | **Para** | usar en cualquier programa de propósito general. |
| **Prioridad** | | Alta. |
| **Duración** | | 0,5 semanas. |
| **Aceptación** | | Se han implementado las operaciones: *flip* vertical, *flip* horizontal, *shear*, escalado, traslación, rotación y transformación afín. Se puede comprobar visualmente que los resultados son correctos en imágenes, puntos, máscaras, mapas de segmentación y mapas de calor. |
| **Tareas** | | ▪ Diseño.<br>▪ Implementación.<br>▪ Pruebas unitarias. |

| **HU.02** - Transformaciones de color | | |
|---|---|---|
| **Descripción** | **Como** | usuario IDALib |
| | **Quiero** | poder hacer transformaciones de color de manera directa sobre imágenes |
| | **Para** | usar en cualquier programa de propósito general. |
| **Prioridad** | | Alta. |
| **Duración** | | 0,5 semanas. |
| **Aceptación** | | Se han implementado las operaciones: cambio de brillo, cambio de contraste, corrección gamma, inyección de ruido, *blur* y ecualización del histograma. Se puede comprobar visualmente que los resultados son correctos. |
| **Tareas** | | ▪ Diseño.<br>▪ Implementación.<br>▪ Pruebas unitarias. |

| **HU.03** - *Pipeline* | | |
|---|---|---|
| **Descripción** | **Como** | usuario IDALib |
| | **Quiero** | poder componer transformaciones y aplicarlas con una cierta probabilidad a lotes de imágenes |
| | **Para** | realizar el *image data augmentation* fijando las operaciones a incluir y probabilidad de aplicar cada una. |
| **Prioridad** | | Alta. |
| **Duración** | | 1 semana. |
| **Aceptación** | | Se ha implementado un *pipeline* que se ejecuta con éxito sobre lotes de imágenes y compone sin errores tanto operaciones geométricas como de color, respetando la probabilidad indicada. |
| **Tareas** | | ▪ Diseño.<br>▪ Implementación de las operaciones geométricas parametrizables para *pipeline*.<br>▪ Composición de las operaciones geométricas.<br>▪ Implementación de las operaciones de color parametrizables para *pipeline*.<br>▪ Composición de operaciones de color.<br>▪ Integración y adaptación a lotes de imágenes.<br>▪ Pruebas unitarias. |

| **HU.04** - Personalización del *pipeline* | | |
|---|---|---|
| **Descripción** | **Como** | usuario de IDALib |
| | **Quiero** | poder personalizar el *pipeline* en aspectos como el modo de interpolación, el modo de extrapolación o el formato de salida |
| | **Para** | realizar el *image data augmentation* adaptado a mis necesidades. |
| **Prioridad** | | Media. |
| **Duración** | | 0,5 semanas. |
| **Aceptación** | | Se puede seleccionar el modo de interpolación, de extrapolación, el formato de salida y el tipo de dato de salida y el comportamiento del *pipeline* cumple los requerimientos. |
| **Tareas** | | ▪ Diseño.<br>▪ Implementación de operaciones con valores aleatorios.<br>▪ Adaptación a varios modos de interpolación.<br>▪ Adaptación a varios modos de extrapolación.<br>▪ Adaptación a varios formatos de salida.<br>▪ Adaptación a la elección del tipo de dato de salida (profundidad de color).<br>▪ Pruebas unitarias. |

| **HU.05** - Visualización de transformaciones individuales | | |
|---|---|---|
| **Descripción** | **Como** | usuario de IDALib |
| | **Quiero** | poder visualizar las transformaciones que realizo |
| | **Para** | depurar mis programas de manera sencilla. |
| **Prioridad** | | Alta. |
| **Duración** | | 1 semana. |
| **Aceptación** | | La herramienta de visualización se despliega con éxito y es validada por el cliente. |
| **Tareas** | | ▪ Diseño.<br>▪ Visualización de imágenes.<br>▪ Visualización de puntos.<br>▪ Visualización de mapas de segmentación y máscaras.<br>▪ Visualización de mapas de calor.<br>▪ Integración interactiva. |

| **HU.06** - Visualización de transformaciones sobre lotes de elementos | | |
|---|---|---|
| **Descripción** | **Como** | usuario de IDALib |
| | **Quiero** | poder visualizar las transformaciones que tienen lugar a través de mi *pipeline* |
| | **Para** | depurar de manera sencilla los programas en los que use el *pipeline* y poder observar la variabilidad de las operaciones. |
| **Prioridad** | | Media. |
| **Duración** | | 0,5 semanas. |
| **Aceptación** | | La herramienta de visualización se despliega con éxito y permite visualizar diferentes elementos del lote. |
| **Tareas** | | ▪ Diseño.<br>▪ Adaptación de la herramienta a lotes de imágenes.<br>▪ Integración. |

| **HU.07** - *Dataloader* | | |
|---|---|---|
| **Descripción** | **Como** | usuario de IDALib |
| | **Quiero** | poder visualizar las transformaciones que tienen lugar a través de mi *pipeline* |
| | **Para** | depurar de manera sencilla los programas en los que use el *pipeline*. |
| **Prioridad** | | Alta. |
| **Duración** | | 1 semana. |
| **Aceptación** | | Se ha implementado el *dataloader* siendo un objeto iterable[1], en el que se puede indicar el *batchsize* (número de elementos que devuelve cada vez que se llama) e implementa el *image data augmentation* de manera correcta. |
| **Tareas** | | ▪ Diseño.<br>▪ Implementación.<br>▪ Pruebas unitarias. |

[1]Un objeto iterable es aquel que permite recorrer sus elementos siguiendo un orden concreto.

| **HU.08** - Implementación del *image data augmentation* a disco | | |
|---|---|---|
| **Descripción** | **Como** | usuario de IDALib |
| | **Quiero** | poder visualizar las transformaciones que tienen lugar a través de mi *pipeline* |
| | **Para** | depurar de manera sencilla los programas en los que use el *pipeline* y ver la variabilidad de las operaciones. |
| **Prioridad** | | Alta. |
| **Duración** | | 1 semana. |
| **Aceptación** | | Se puede generar un *dataset* aumentado a disco, indicando el tamaño del *dataset* de salida. Se guarda correctamente la salida en disco, pudiendo comprobar visualmente que el *image data augmentation* se ha realizado de manera correcta. |
| **Tareas** | | ▪ Diseño.<br>▪ Implementación.<br>▪ Pruebas unitarias. |

| **HU.09** - Estandarización del código (PEP8) | | |
|---|---|---|
| **Descripción** | **Como** | usuario de IDALib |
| | **Quiero** | poder leer de forma clara el código fuente |
| | **Para** | entenderlo mejor y, si es el caso, poder ampliarlo de manera sencilla. |
| **Prioridad** | | Media. |
| **Duración** | | $<$ 0,5 semanas. |
| **Aceptación** | | El código cumple con las buenas prácticas establecidas en la guía PEP8 en lo que respecta a su formato. |
| **Tareas** | | ▪ Estandarización del código siguiendo la guía de buenas prácticas PEP8. |

| **HU.10** - Generación de la documentación de la librería | | |
|---|---|---|
| **Descripción** | **Como** | usuario de IDALib |
| | **Quiero** | poder acceder de manera sencilla a la documentación de la librería |
| | **Para** | agilizar el proceso de aprendizaje de la librería y depurar los programas de forma sencilla. |
| **Prioridad** | | Media. |
| **Duración** | | $< 0{,}5$ semanas. |
| **Aceptación** | | Se ha creado una documentación que recoge los aspectos esenciales sobre la librería (instalación, descripción y documentación de los métodos). |
| **Tareas** | | ▪ Generación de la documentación de la librería. |

| **HU.11** - Publicación de la librería y documentación | | |
|---|---|---|
| **Descripción** | **Como** | usuario de IDALib |
| | **Quiero** | poder descargar directamente la librería como cualquier otra librería estándar |
| | **Para** | evitar la lectura de guías o tutoriales para realizar su instalación. |
| **Prioridad** | | Baja |
| **Duración** | | $< 0{,}5$ semanas. |
| **Aceptación** | | Tanto la librería como la documentación son accesibles en repositorios públicos. |
| **Tareas** | | ▪ Publicación de la librería en el repositorio PyPI.<br>▪ Publicación de la documentación de la librería en ReadTheDocs. |

## 4.2. Análisis de herramientas

Además de analizar las necesidades funcionales, es importante analizar qué herramientas usaremos para llevar a cabo el proyecto. En esta sección se detallan las herramientas y tecnologías seleccionadas para el desarrollo del proyecto y se justifica su elección.

Las herramientas empleadas como plataforma de desarrollo para diferentes elementos del proyecto fueron:

- **PyCharm**: IDE empleado para el desarrollo de todo el proyecto, ya que está enfocado al desarrollo en Python e incluye herramientas útiles para depurar y analizar el código desarrollado.
- **Python 3.6**: lenguaje de programación de la librería.
- **Pytest**: librería utilizada para las pruebas. A pesar de no ser la opción estándar para Python, permite la organización de las pruebas de forma clara eliminando duplicados de código.
- **ProjectLibre y WBSTool**: herramientas usadas para la planificación y durante el seguimiento para el diseño del EDT y la asignación de horas.
- **StarUML y Draw.io**: editores de diagramas usados en las fases de diseño y documentación en la memoria.
- **Git y Github**: herramientas fundamentales para el control de versiones de la librería.

Las herramientas empleadas de forma interna en la librería suponen una decisión más importante ya que pueden afectar de forma directa al rendimiento del proyecto. A continuación detallamos cada una de ellas:

- **Librería de *Deep Learning*:** entre las librerías más populares de *deep learning* encontramos Keras, Tensorflow y Pytorch. Keras es muy sencilla de usar, pero presenta ciertas limitaciones, mientras que Tensorflow presenta una curva de aprendizaje demasiado elevada. **Pytorch** ofrece un buen compromiso entre la flexibilidad y la facilidad de uso, por lo que es la seleccionada para el proyecto. IDALib se diseña para integrarse con sistemas de *deep learning* programados en esta librería.
- **Librería para transformaciones geométricas a bajo nivel**: estas transformaciones son las más complejas a nivel de procesado computacional, por ello la decisión de qué librería emplear es muy importante para el proyecto. En la sección 5.1.2 se detalla la investigación llevada a cabo para esta elección, en la que se concluye con la selección de **Kornia**.
- **Librería para transformaciones de color a bajo nivel**: estas transformaciones también tienen un impacto importante en el rendimiento del sistema. Por ello también se llevó a cabo una investigación detallada que se describe en la sección 5.1.2. Como conclusión, la librería seleccionada fue **OpenCV**.
- **Librería para la herramienta de visualización**: entre los paquetes destinados a la visualización de datos en Python, el más común por su sencillez

es **matplotlib**, pero su alcance es limitado para la realización de una visualización interactiva. Existen otras opciones más avanzadas, de las cuales, **Bokeh** fue la escogida al ser de propósito más general y estar enfocada a la creación de visualizaciones interactivas. Además, al estar basado en HTML y Javascript, su curva de aprendizaje no es muy elevada.

# Capítulo 5

# Diseño e implementación

En este capítulo se detallan las fases de diseño e implementación del producto. Al emplear Scrum, tanto el diseño como la implementación y parte del análisis del producto se realizaron iterativamente a lo largo del desarrollo del proyecto.

Este capítulo se divide en los diferentes *sprints* de desarrollo, detallando en cada uno de ellos los *spikes* o historias de usuario. A lo largo de este capítulo se van a referenciar diferentes épicas, *spikes* e historias de usuario que ya han sido detalladas en la sección 4.1 (*product backlog*).

## 5.1. Sprint 1

### 5.1.1. *Sprint Backlog*

En este primer *sprint* era necesario tener suficientes conocimientos las transformaciones de imágenes para poder diseñar de manera adecuada la librería. Se decide empezar por 2 *spikes* de formación, tanto en transformaciones geométricas como en transformaciones de color.

Se estimó que ambas se podían llevar a cabo en 1 semana de duración, con 0,5 semanas para cada *spike*. Se decidió hacer ambos *spikes* de manera sucesiva, empezando por las transformaciones geométricas (ya que estas representan mayor complejidad computacional y por lo tanto tienen más prioridad para el proyecto).

Además de la formación, era necesaria la elección de las librerías a bajo nivel para la implementación de la librería objeto de este proyecto (tanto para las operaciones geométricas como las de color).

Como cada herramienta emplea estructuras y tipos de datos diferentes, lo que puede ser un problema para integrarlas en una misma librería, se decidió seleccionar como máximo 2 librerías de bajo nivel, separando las transformaciones a nivel de píxel de las transformaciones espaciales.

Se añade un tercer *spike* al *sprint* que consistirá en un estudio de rendimiento e investigación sobre las mejores alternativas. Se estima una duración de una semana para llevar a cabo la investigación.

| *Sprint* Backlog | |
|---|---|
| **Nº de *sprint*** | *Sprint* 1 |
| **Objetivo** | Obtener la formación teórica suficiente para tomar las decisiones de diseño adecuadas. Realizar las pruebas de rendimiento necesarias para decidir qué herramientas de bajo nivel son las adecuadas para la implementación de las transformaciones. |
| **Duración** | 2 semanas. |
| **Items asignados** | 1. **SPK.01** - Formación en transformaciones espaciales (geométricas) (0,5 semanas).<br>2. **SPK.02** - Formación en transformaciones a nivel de píxel (de color) (0,5 semanas).<br>3. **SPK.03** - Estudio de rendimiento de librerías (1 semana). |

### 5.1.2. Estudio de rendimiento

A continuación se detallan las pruebas llevadas a cabo para seleccionar la mejor librería de bajo nivel para el proyecto. En primer lugar, se estudia la librería para transformaciones geométricas ya que al ser operaciones más costosas a nivel de rendimiento, tiene más importancia para el proyecto. Una vez determinada, se realiza la investigación para las transformaciones de color.

Para el desarrollo de la librería interesa emplear las implementaciones de las transformaciones geométricas más eficientes y rápidas. No tiene sentido realizar la implementación propia de estas operaciones, ya que existen herramientas optimizadas para su funcionamiento a bajo nivel, y así obtenemos mayor flexibilidad en la implementación del proyecto.

Pese que existe gran variedad de opciones, se han seleccionado las siguientes por su uso extendido y por aquellas cualidades diferenciadoras de cada una:

- **OpenCV**: librería de código abierto de visión por computador originalmente desarrollada por Intel. Destaca por ser un proyecto sobresaliente en el ámbito, cuenta con versiones en CPU y GPU y su implementación basada en C++ que aprovecha las características de los procesadores para proporcionar un buen rendimiento.

- **Kornia**: librería de código abierto que consiste en un conjunto de rutinas y módulos diferenciables para resolver problemas genéricos de visión por computador. Es un proyecto inspirado en OpenCV pero que destaca por sus operaciones diferenciables [1].

- **NVIDIA DALI** (Data Loading Library): librería de código abierto para decodificar y aumentar imágenes y vídeos orientada a acelerar las aplicaciones de aprendizaje profundo. Es una librería relativamente reciente muy orientada al máximo rendimiento en el preprocesado de datos. Destaca por ser un proyecto propio de la compañía de GPUs NVIDIA, lo que proporciona altas prestaciones.

- **Scikit-Image**: librería conformada como una colección de algoritmos para el procesado de imágenes de código abierto de uso muy extendido. Es la librería estándar de tratamiento de imágenes en Python.

Han quedado descartadas otras opciones en base a la experimentación realizada por la comunidad: por ejemplo la librería Fastai[2] no se evalúa en base a un experimento[3] en el que se refleja que NVIDIA DALI tiene un rendimiento del orden de 5 veces superior.

Para evaluar el rendimiento de cada librería se medirá el tiempo de ejecución de cada una sobre conjuntos con distinto número de imágenes. Se emplean imágenes para el experimento y no los otros tipos de datos considerados ya que, no todas ellas incluyen soporte para los metadatos considerados.

Para evaluar de forma correcta el rendimiento e las distintas librerías, es importante evaluar la composición de operaciones ya que las operaciones individuales no son comunes en el *image data augmentation*, sino que por norma general se aplica una composición de ligeras transformaciones a cada elemento del *dataset*. Además, para evaluar la escalabilidad de cada herramienta, se realizarán pruebas con diferentes números de imágenes.

Por todo lo anterior, las condiciones seleccionadas para realizar las medidas son:

- **Composición de transformaciones**: transformación afín expresada por matriz + *flip*.
  Se emplea una transformación afín ya que mediante la matriz adecuada se puede expresar cualquiera de las transformaciones geométricas abordadas. No obstante, como todas las librerías incluyen operaciones independientes, se incluye también una operación de *flip* posterior (ya que está presente en todas las opciones y no depende de parámetros que puedan ser diferentes

[1] Operaciones que admiten derivación y por tanto, pueden incluirse dentro de una red neuronal artificial.

[2] https://www.fast.ai/

[3] https://github.com/slawekslex/dl_experiments/blob/master/misc/DALIExperiment.ipynb

entre las librerías) para evaluar las transformaciones independientes y poder valorar si se aborda de manera eficiente la composición de operaciones.

- **Número de imágenes**: 1, 4, 8, 20, 50, 100, 400
  Se aplicarán las transformaciones en lotes de diferente número imágenes para evaluar el comportamiento de cada opción y sus posibles optimizaciones y paralelizaciones.
- **Dimensiones de la imagen**: 250 x 250 x 1 (en escala de grises), seleccionada como valor de compromiso genérico.
- **Número de pruebas**: para cada medida se realizan 20 pruebas idénticas, y se calcula el promedio de las 20 medidas.

A continuación se describen los experimentos que recogen todas estas consideraciones:

1. **OpenCV:** se realizan 3 tipos de experimentos:
   - OpenCV sobre CPU.
   - OpenCV sobre CPU con *multithread* (ejecución en paralelo en varios hilos).
   - OpenCV sobre GPU (empleando objetos `uMat` y la directiva `device.to(cuda)`.

   Para evaluar la librería se ejecuta la composición de transformaciones de OpenCV (`flip` y `WarpAffine`) sobre lotes de `arrays numpy`.
2. **Kornia**: se evalúan dos versiones de la librería:
   - Kornia sobre CPU.
   - Kornia sobre GPU (cargando los tensores de la operación en GPU con el método `tensor.to(cuda)`.

   Para evaluar la librería, se pasan lotes de tensores `torch`[4] por un *pipeline* de Kornia, que aplica las transformaciones de `flip` y `WarpAffine`.
3. **Scikit-image:** se evalúan dos versiones de la librería:
   - Scikit-image.
   - Scikit-image con *multithreading* (ejecución en paralelo en varios hilos).

   Para evaluar la librería se ejecuta una operación compuesta que incluye la operación afín y el *flip*.

[4]Los tensores son una generalización de las matrices y pueden entenderse como arrays de $n$ dimensiones.

4. **NVIDIA DALI**: esta herramienta solo incluye soporte para GPU, por lo que solo se evalúa esta versión. Emplea tipos de datos propios y optimizaciones complejas que hacen más complicada la realización de la experimentación, en la que hay que tener especial cuidado con las optimizaciones de la librería. Para definir las operaciones se construye un *pipeline* que, en esta librería, está optimizado para procesarse de manera no síncrona. Para realizar una medición real del tiempo que requieren las transformaciones fueron necesarias las siguientes acciones:

   - Evitar ejecuciones no síncronas y concurrentes:

     ```
     exec_async=False,
     # evitar que la llamada a pipeline.run ejecute de manera
     # asíncrona el thread del pipeline

     exe_pipelined=False
     #evita la ejecución concurrente de cpu y gpu
     ```

   - Definir las transformaciones del *pipeline*:

     ```
     # Affine
     self.augmentations["warpaffine"] = ops.WarpAffine(
     device = "gpu",
     matrix = [1.0, 0.4, 0.0, 0.0, 1.2, 0.0])

     # Flip
     self.augmentations["flip"] = ops.Flip(device = "gpu",
     vertical = 1,
     horizontal = 0)
     ```

En el Cuadro 5.1 se muestran los valores obtenidos con la experimentación descrita anteriormente, que se representan gráficamente en la Figura 5.1. En ella se observa que el escalado de todas las librerías con el número de imágenes sigue la misma tendencia. Destaca Kornia sobre GPU, ya que su rendimiento aumenta notablemente con tamaños mayores de lote (aunque con tamaños pequeños no da los mejores resultados). Con este primer experimento es evidente que hay gran diferencia entre las librerías, presentando claramente un peor rendimiento temporal opciones como Scikit-image o Kornia sobre CPU.

Cuadro 5.1: Medidas de tiempo de transformación de cada librería según el número de imágenes. En verde, la mejor medida para cada valor de *batchsize*.

| Batchsize / Library | 1 | 4 | 8 | 20 | 50 | 100 | 400 |
|---|---|---|---|---|---|---|---|
| **Kornia CPU** | 0,0082 | 0,0296 | 0,0167 | 0,0347 | 0,0789 | 0,1470 | 0,6189 |
| **Kornia GPU** | 0,0055 | 0,0183 | 0,0089 | 0,0099 | 0,0101 | 0,0125 | 0,0483 |
| **OpenCV CPU** | 0,0007 | 0,0020 | 0,0042 | 0,0103 | 0,0256 | 0,0510 | 0,2114 |
| **OpenCV CPU MT** | 0,0007 | 0,0011 | 0,0022 | 0,0041 | 0,0096 | 0,0216 | 0,0845 |
| **OpenCV GPU** | 0,0003 | 0,0016 | 0,0019 | 0,0044 | 0,0104 | 0,0208 | 0,0940 |
| **NVIDIA DALI** | 0,0013 | 0,0013 | 0,0014 | 0,0023 | 0,0049 | 0,0076 | 0,0261 |
| **Scikit image** | 0,0039 | 0,0094 | 0,0186 | 0,0462 | 0,1154 | 0,2305 | 0,9380 |
| **Scikit image (MT)** | 0,0025 | 0,0036 | 0,0051 | 0,0120 | 0,0280 | 0,0532 | 0,2305 |

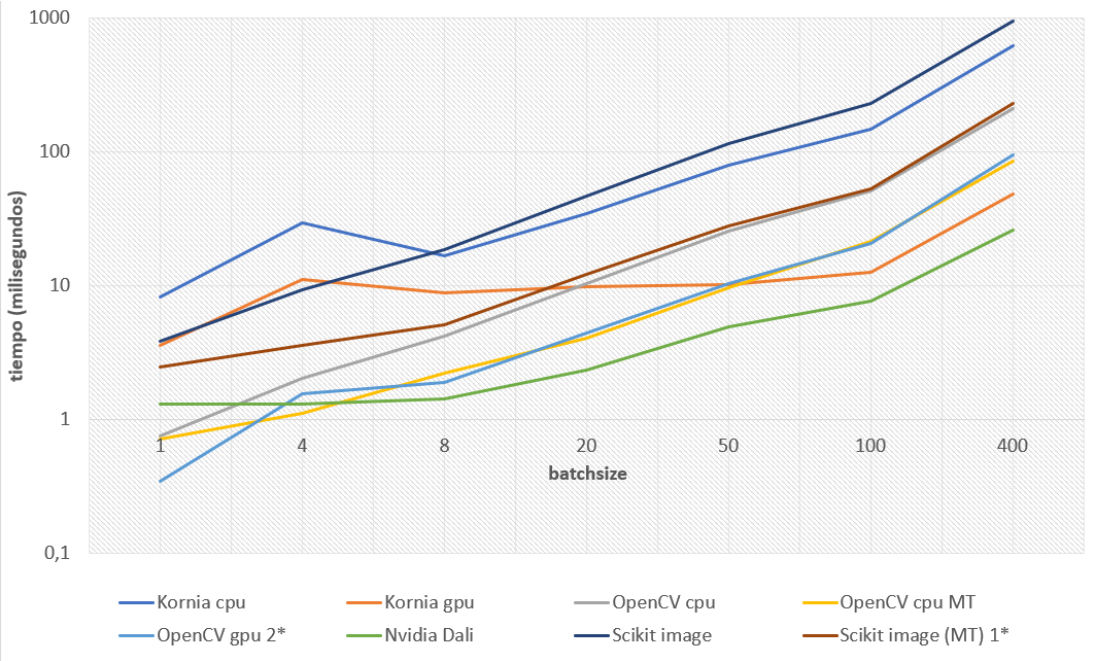



Figura 5.1: Representación gráfica de las medidas de tiempo de las librerías según el número de imágenes (escala logarítmica).

Para tener una referencia de comparación (en tiempo de ejecución) entre las librerías, se toman las medidas sobre 400 imágenes (ya que hay una mayor diferenciación), y se construye la Figura 5.2. En ella se muestra la proporción de tiempo que necesita cada librería con respecto al menor tiempo medido (el de NVIDIA DALI). Utilizando estos valores, por orden de mejor a peor rendimiento, las opciones se posicionan de la siguiente manera:

1. **NVIDIA DALI**: mejor rendimiento.

2. **Kornia sobre GPU**: 1,8 veces más lento que NVIDIA DALI.
3. **OpenCV sobre CPU con multithread**: 3,2 veces más lento que NVIDIA DALI.
4. **OpenCV sobre GPU**: 3,6 veces más lento que NVIDIA DALI.
5. **OpenCV sobre CPU**: 8,1 veces más lento que NVIDIA DALI.
6. **Scikit-image con multithread**: 8,8 veces más lento que NVIDIA DALI.
7. **Kornia sobre CPU**: 23,7 veces más lenta que NVIDIA DALI.
8. **Scikit-image**: 35,9 veces más lenta que NVIDIA DALI.

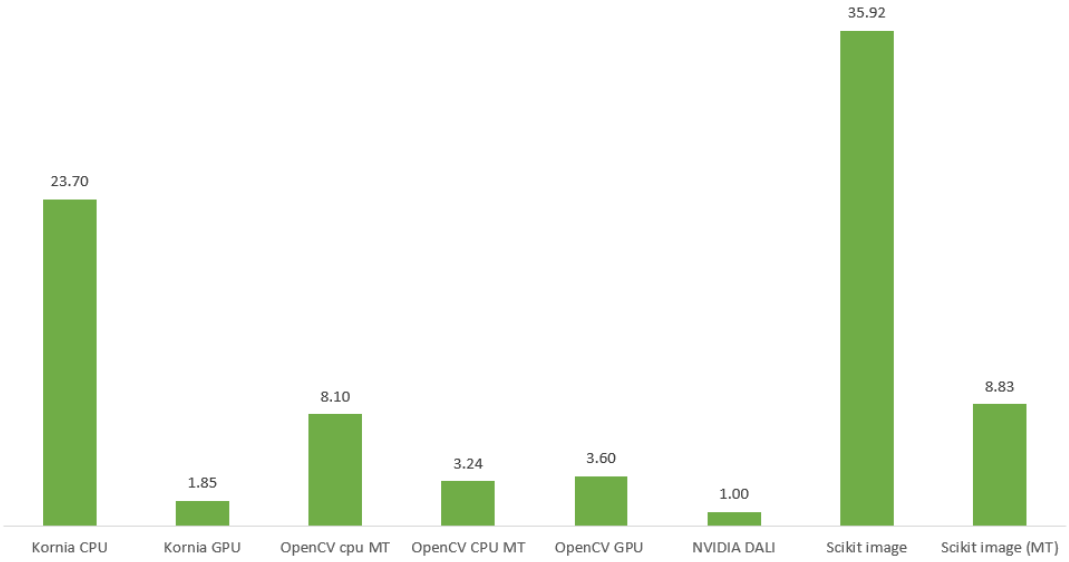



Figura 5.2: Tiempo proporcional de cada librería respecto el menor tiempo medido para 400 imágenes.

Después de este experimento, las 4 mejores opciones son NVIDIA DALI, Kornia sobre GPU, OpenCV sobre CPU con multithread y OpenCV sobre GPU, pero necesitaremos tener en cuenta otros factores para realizar la selección final de la librería. Para ello, se evalúa de manera más exhaustiva el rendimiento en función de otros factores:

1. **Resolución de la imagen**: se mide el rendimiento para las siguientes resoluciones:
   - 50 x 50

- 100 x 100
- 250 x 250
- 500 x 500
- 1500 x 1500

2. **Número de canales de la imagen**: se compara el rendimiento con los siguientes números de canales:

   - 1
   - 3
   - $10^5$
   - 50
   - 100
   - 1000

   Ambos experimentos se realizan sobre lotes de 100 imágenes de 1 canal y resolución 100x100.

Para esta segunda fase de experimentación, se emplean las mismas estructuras de datos y de transformaciones detalladas en el apartado anterior. Los resultados obtenidos son los que se muestran en los Cuadros 5.2 y 5.3 para distintas resoluciones y canales, respectivamente.

Cuadro 5.2: Tiempo (milisegundos) de transformación por imagen con diferentes resoluciones de imagen.

| **Resolución** / **Librería** | **50x50** | **100x100** | **250x250** | **500x500** | **1500x1500** |
|---|---|---|---|---|---|
| **OpenCV mt** | 0,110 | 0,120 | 0,206 | 0,596 | 4,473 |
| **OpenCV GPU** | 0,504 | 0,200 | 0,204 | 0,338 | 2,476 |
| **Kornia GPU** | 0,127 | 0,086 | 0,109 | 0,202 | 1,072 |
| **NVIDIA DALI** | 0,065 | 0,050 | 0,079 | 0,203 | 0,989 |

[5]Pese que no existen imágenes de más de 4 canales, es importante evaluar el comportamiento con números de canales mayores a 4 para valorar la concatenación de varias imágenes en las transformaciones

Cuadro 5.3: Tiempo (en milisegundos) de transformación por imagen y por canal con diferente número de canales de imagen.

| Librería \ Canales | 1 | 3 | 10 | 100 | 1000 |
|---|---|---|---|---|---|
| **OpenCVmt** | 0.120 | 0.110 | 0.088 | 0.111 | 0.157 |
| **OpenCV GPU** | 0.2000 | 0.411 | 0.980 | 8.851 | 29.978 |
| **Kornia GPU** | 0.086 | 0.070 | 0.058 | 0.065 | 0.094 |
| **NVIDIA DALI** | 0.050 | 0.016 | 0.031 | 0.029 | 0.045 |

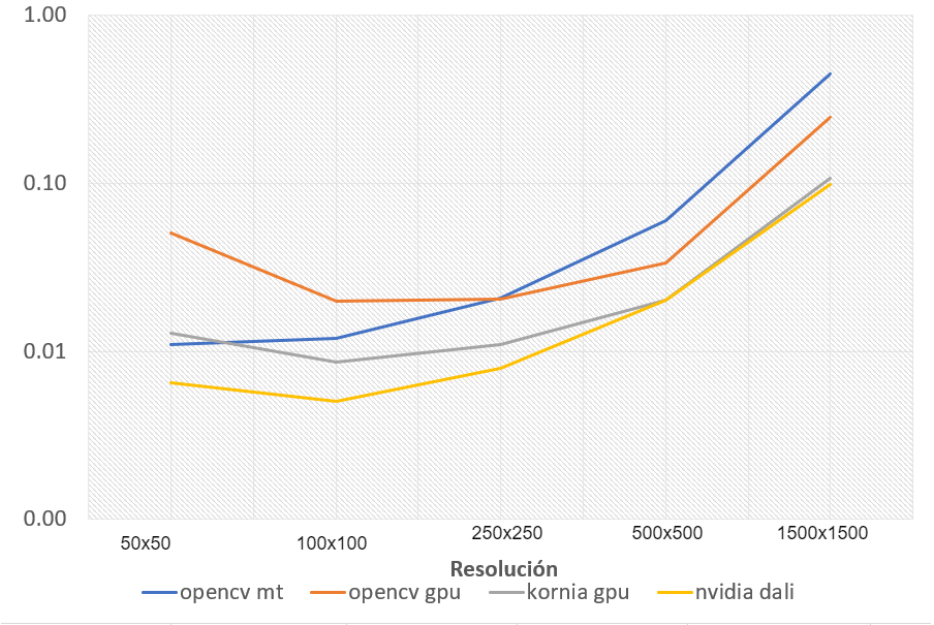



Figura 5.3: Representación gráfica del tiempo de transformación de las librerías en función de la resolución de la imagen.

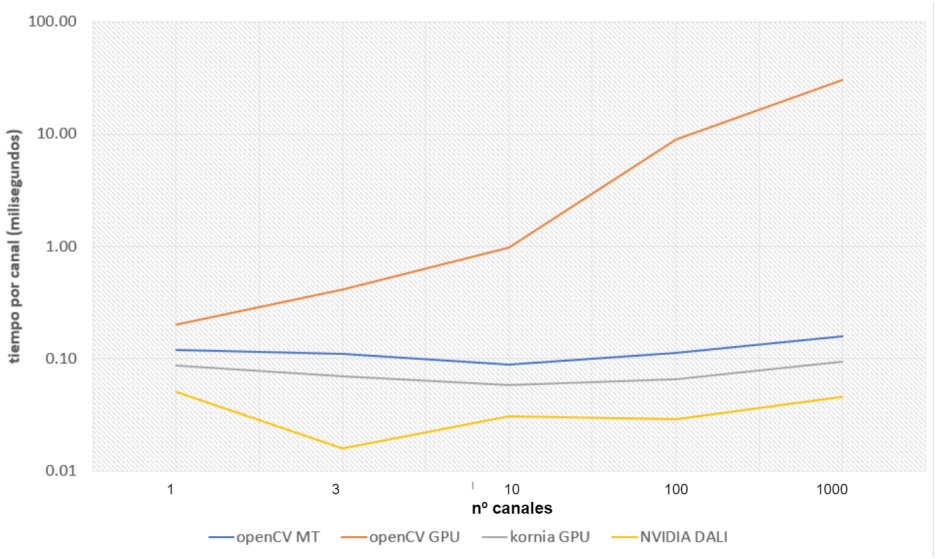



Figura 5.4: Representación gráfica del tiempo de transformación de las librerías por canal variando el número de canales de la imagen.

Tanto en lo que refiere a la resolución de la imagen (Figura 5.3) como en lo que refiere al número de canales (Figura 5.4), la mejor opción está disputada entre NVIDIA DALI y Kornia en GPU. En general, NVIDIA DALI parece más rápida. No obstante, atendiendo al Cuadro 5.2, parece que Kornia GPU tiende a equipararla en tamaños de imagen mayores. Es decir, Kornia en GPU escala mejor con el tamaño de la imagen.

Con el número de canales es interesante observar como la opción que peor rendimiento muestra es OpenCV sobre GPU. En general la tendencia parece indicar que a partir de 10 canales el rendimiento empieza a bajar.

Con estos datos podemos extraer las siguientes conclusiones:

- La librería con mejor rendimiento es NVIDIA DALI pero:
  - Es una librería muy cerrada y centrada en su propio objetivo, por lo que puede resultar complicado tratar de adaptarla a nuestro proyecto.
  - Las opciones que ofrece son limitadas y la curva de aprendizaje al usar tipos de datos propios y funciones de bajo nivel en C++ es elevada.
- OpenCV con multithread muestra un buen rendimiento, pero el requerimiento de memoria que suponen los hilos es elevado, además de no conseguir alcanzar el rendimiento de Kornia.
- Kornia ofrece muy buenos resultados y operaciones flexibles que pueden ser fácilmente integrables en otras herramientas.
- OpenCV en GPU ofrece buenos resultados pero no consiguen alcanzar los de Kornia.

Por todo ello, se seleccionó **Kornia sobre GPU** para la implementación de las transformaciones geométricas. Esta librería trabaja directamente con tensores `torch` (tipo de dato de entrada a las redes neuronales diseñadas con Pytorch), lo cual se adapta muy bien a las necesidades de nuestra librería, ya que facilita su integración con proyectos de aprendizaje profundo con Pytorch.

Una vez elegida la librería para las transformaciones geométricas, tendremos que analizar las librerías a utilizar para las transformaciones de color. A la hora de elegir una librería para este propósito, el factor eficiencia cobra una menor importancia, ya que en general no se incluyen operaciones pesadas a nivel de procesado que impliquen multiplicaciones matriciales o interpolación. Las operaciones de color solo afectan a la imagen (de todos los posibles tipos de datos de entrada), lo que reduce aún más su impacto en el rendimiento. Por todo esto, toman más importancia para la decisión aspectos como:

- El tipo de dato con el que se trabaja.
- La integración de esta librería con la seleccionada para las transformaciones geométricas.
- La repertorio de operaciones que ofrece.
- El soporte para Look-Up-Tables (LUTs).

El Cuadro 5.4 muestra un resumen de las operaciones de color integradas en las distintas librerías. En él vemos que no hay grandes diferencias en el repertorio de operaciones de color pero, como consideramos de gran utilidad el soporte para LUTs, las librerías que no incluyen su soporte (Kornia y Scikit-image) quedan descartadas. Entre las que sí dan soporte para LUTs (OpenCV y NVIDIA DALI), podemos ver que OpenCV cuenta con más variedad de operaciones. Además, como hemos indicado anteriormente, NVIDIA DALI es una herramienta compleja y por tanto difícil de adaptar a las necesidades de nuestra librería.

Cuadro 5.4: Tabla comparativa de operaciones de color ofrecidas por cada librería.

| | **Kornia** | **OpenCV** | **NVIDIA DALI** | **Scikit-image** |
|---|---|---|---|---|
| **LUT** | No | Si | Si | No |
| ***Blur*** | Si | Si | No | Si |
| ***Brightness*** | Si | Si | Si | Si |
| ***Contrast*** | Si | Si | Si | Si |
| ***Gamma*** | Si | Si | No | Si |
| ***Inject noise*** | No | No | No | Si |
| ***Equalization*** | Si | Si | No | Si |

Todos los datos y explicaciones indicados en este apartado parecen suficiente para elegir OpenCV. Sin embargo, esta librería trabaja con datos de tipo `array numpy`, mientras que Kornia trabaja con tensores. Esto no tiene por qué ser un problema sabiendo que la lectura de imágenes por norma general se hace con `array numpy` mientras que Pytorch trabaja con `tensores torch`. Como ambos tipos de datos son necesarios, interesa que la herramienta trabaje con ambos.

Por lo tanto, y como conclusión de este *sprint*, la librería seleccionada para la implementación de las transformaciones geométricas a bajo nivel es **OpenCV**.

## 5.2. *Sprint* 2

Una vez elegidas las librerías de bajo nivel para la implementación, se analizan las siguientes tareas del *product backlog*. Se pueden identificar las siguientes dependencias:

1. El *pipeline* necesita la implementación de las operaciones a bajo nivel para poder comenzar su implementación.

2. El módulo *dataloader* de entrada para la red neuronal requiere la implementación del *pipeline*.

3. El módulo de *image data augmentation* a disco requiere tener la implementación del *pipeline*.

### 5.2.1. *Sprint Backlog*

Lo que más prioridad tiene para el proyecto es la implementación de las transformaciones independientes. Como las librerías seleccionadas son diferentes para las transformaciones geométricas y para las transformaciones a nivel de píxel se divide la épica inicial (**FTR.01** Operaciones de transformaciones) en 2 historias de usuario: (**HU.01**: Transformaciones geométricas y **HU.02**: Transformaciones de color).

A este segundo *sprint* se le asignan estas 2 historias de usuario, que se estima que se pueden completar en 1 semana de duración (0,5 semanas cada una).

| *Sprint* Backlog | |
|---|---|
| **Nº de *sprint*** | *Sprint* 2 |
| **Objetivo** | Implementar el primer módulo de la librería que compone las transformaciones individuales sobre los elementos en base a la investigación realizada en el *sprint* anterior. |
| **Duración** | 1 semana. |
| **Items asignados** | 1. **HU.01**: Transformaciones geométricas. (0,5 semanas)<br>2. **HU.02**: Transformaciones de color (0,5 semanas) |

### 5.2.2. Análisis

Para la implementación de estas operaciones se identifican los siguientes requisitos:

- Al emplear imágenes de tipo `numpy` y de tipo `torch tensor` en las herramientas a bajo nivel se decide que todas las transformaciones deben admitir estos 2 tipos de datos de entrada y mantenerlos en la salida.
- Todas las operaciones deben admitir elementos de entrada compuestos por cualquier combinación de los siguientes datos:
  - Imágenes: tanto en escala de grises como en color.
  - Máscaras.
  - Mapas de segmentación.
  - Mapas de calor.
  - Coordenadas de puntos.
- Debe admitir cualquier combinación de esos tipos de datos, incluyendo varios datos de un mismo tipo (por ejemplo que incluya 2 máscaras diferentes o $n$ mapas de calor).
- Las operaciones deben respetar la profundidad de color de los elementos de entrada.
- Para ofrecer suficiente diversidad y basándonos en estudios sobre estrategias de *image data augmentation* [11, 13, 14, 15] se seleccionaron las siguientes **operaciones para incluir en la librería**:
  - Transformaciones espaciales:

  - Rotación: debe permitir indicar el centro de rotación.
  - *Shear* (estiramiento).
  - Traslación.
  - Operación afín.
  - *Flip* (reflexión) vertical.
  - *Flip* (reflexión) horizontal.
  - Escalado: debe permitir indicar el centro de escalado.
- Transformaciones a nivel de píxel:
  - Cambio de brillo.
  - Cambio de contraste.
  - Corrección gamma.
  - Ecualización del histograma.
  - Inyección de ruido.
  - Desenfoque (*blur*).

### 5.2.3. Diseño

En este apartado se detalla el diseño y decisiones tomadas para la implementación del módulo de transformaciones geométricas:

Como se mencionó en los requisitos, el dato de entrada para las operaciones puede incluir cualquier combinación de los objetos bidimensionales identificados. Por ello se decidió emplear un **diccionario Python** como tipo de dato de entrada. Cada uno de los elementos que incluya se asociará al tipo de dato que represente (*image*, *mask*, *segmap*, *heatmap* o *keypoints*). También debe permitir la entrada de elementos no susceptibles a la transformación, por ejemplo la etiqueta de la imagen. Todos los elementos cuyo nombre no contenga ninguna de las palabras clave permanecerán intactos en la transformación. Por ejemplo:

```
{"image" : img, "mask" : mask, "mask2" : mask_2, "mask3" : mask_3,
"keypoints" : points, "target" : 35}
```

La transformación actuará sobre 1 imagen, 3 máscaras y una lista de coordenadas de puntos, manteniendo el *target* intacto. De este modo se consigue un tipo de dato compacto al que no se tendrá que modificar la estructura para aplicarle la transformación. Los elementos a transformar pasarán por la transformación de manera completa, encargándose el sistema de detectar el tratamiento que debe dar a cada metadato. Este formato de dato será el empleado por toda la librería en sus diferentes módulos.

Es importante tener en cuenta el tratamiento especial que se le debe dar a cada tipo de dato:

- **Imágenes y mapas de calor**: tratamiento normal de las librerías de bajo nivel.
- **Máscaras y mapas de segmentación**: estos tipos de datos se caracterizan por sus valores discretos. En las operaciones geométricas por resultado de la interpolación se pueden generar valores intermedios. Con estos datos se debe emplear siempre el método de interpolación "vecino más cercano" (*nearest*) que, al tomar el valor más cercano, nunca genera valores intermedios.
- **Coordenadas de puntos**: para la aplicación de transformaciones geométricas a las coordenadas de puntos, éstos se transforman a coordenadas homogéneas y se multiplican por la matriz de transformación que corresponda.

El esquema de funcionamiento del módulo es el representado en la Figura 5.5. El tipo de dato de entrada es el diccionario Python ya explicado. Éste va a ser procesado de la siguiente manera antes de aplicar la transformación:

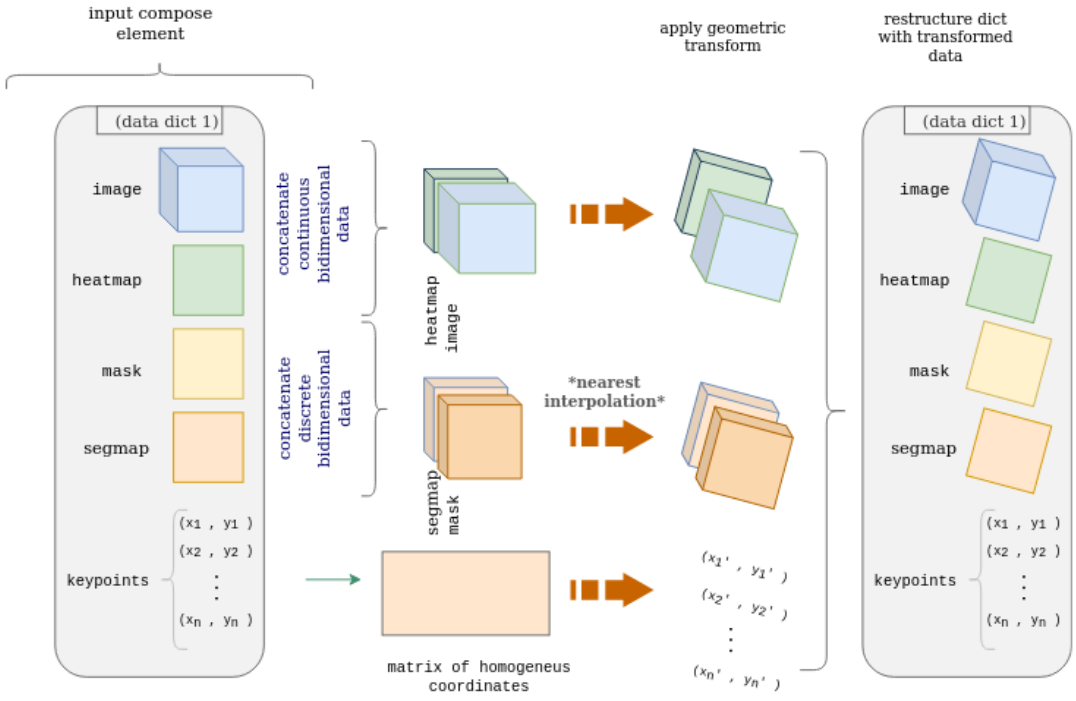



Figura 5.5: Esquema del módulo de transformaciones geométricas.

1. Todas las imágenes y mapas de calor se concatenan en un único elemento a lo largo del tercer eje.
2. Todos los mapas de segmentación y máscaras de entrada se concatenan de la misma manera en un 2º elemento interno.

3. Las coordenadas de puntos se transforman a su correspondiente coordenadas homogéneas y se disponen en una única matriz como se ve en la ecuación 5.1, que tendrá tantas columnas como puntos de entrada:

$$\begin{matrix}(x_1, y_1)\\(x_2, y_2)\\...\\(x_n, y_n)\end{matrix} \longrightarrow \begin{pmatrix}x_1 & x_2 & ... & x_n\\y_1 & y_2 & ... & y_n\\1 & 1 & ... & 1\end{pmatrix} \tag{5.1}$$

4. Los 3 elementos se convierten (si no lo son ya) en `tensores torch` y se cargan en GPU.

Una vez aplicado este preprocesado de los datos, se aplica la transformación geométrica correspondiente sobre cada uno de ellos. De esta manera, independientemente del número de elementos de entrada, se aplican como máximo:

1. **Una transformación** sobre el elemento compuesto por imágenes y mapas de calor.

2. **Una transformación con interpolación de tipo vecino más cercano** sobre el elemento compuesto por las máscaras y los mapas de segmentación.

3. Una **multiplicación matricial**: al disponer los puntos en la matriz se puede aplicar la transformación como una única multiplicación matricial (ecuación 5.2), donde la primera matriz representa la matriz de transformación, la segunda la matriz de coordenadas homogéneas y el resultado obtenido es una matriz con las coordenadas transformadas.

$$\begin{pmatrix}t_{1,1} & t_{1,2} & t_{1,3}\\t_{2,1} & t_{2,2} & t_{2,3}\\t_{3,1} & t_{3,2} & t_{3,3}\end{pmatrix} \times \begin{pmatrix}x_1 & x_2 & ... & x_n\\y_1 & y_2 & ... & y_n\\1 & 1 & ... & 1\end{pmatrix} = \begin{pmatrix}x_{t1} & x_{t2} & ... & x_{tn}\\y_{t1} & y_{t2} & ... & y_{tn}\\ & & ... & \end{pmatrix} \tag{5.2}$$

Finalmente se realiza un post-procesado de los datos, dividiendo los elementos concatenados en sus respectivas clases originales, y restaurando la matriz de coordenadas de puntos a su formato original para devolver el mismo formato del diccionario de entrada.

### Transformaciones a nivel de píxel

Las transformaciones a nivel de píxel, al actuar únicamente sobre imágenes, resultan más sencillas. La entrada para las transformaciones sigue el mismo formato tipo diccionario Python ya explicado. Sobre él, lo único que es necesario es:

- Identificar aquellos elementos que sean de tipo imagen.

- Como la librería a bajo nivel empleada es OpenCV, los datos en este caso se transforman (si no lo son ya) a tipo `numpy`.
- Se aplica la transformación correspondiente.
- Se vuelve a convertir la imagen al tipo de dato original (si se ha modificado) y se devuelve el diccionario de entrada ya transformado.

Las operaciones de cambio de brillo, cambio de contraste, corrección gamma y ecualización no admiten variaciones, pero en el caso de la inyección de ruido y el desenfoque existen diferentes aproximaciones para llevarlo a cabo:

- **Inyección de ruido**: se implementan las 4 aproximaciones más comunes:
  - ***Salt and pepper***: tipo de ruido estadístico compuesto de píxeles blancos (sal) y negros (pimienta).
  - **Ruido gaussiano**: tipo de ruido estadístico que tiene una función de densidad de probabilidad (PDF) igual a la de la distribución normal [22].
  - **Ruido moteado o *Speckle***: una interferencia granular que existe intrínsecamente en las imágenes de radiografías y que degrada su calidad [21].
  - **Ruido *Poisson***: tipo de ruido estadístico que tiene una función de densidad de probabilidad (PDF) igual a la de la distribución de Poisson.
- **Desenfoque**:
  - **Desenfoque de caja**: el valor de cada píxel se calcula promediando el valor de los píxeles que lo rodean.
  - **Desenfoque gaussiano**: el valor de cada píxel se calcula aplicando una función gaussiana a los píxeles que lo rodean.

### 5.2.4. Implementación

A continuación se detallan brevemente las estructuras empleadas en la implementación. Tanto en las transformaciones geométricas como en las transformaciones de color, para las tareas de pre y post-procesado de los datos se emplean decoradores Python[6].

En concreto, en las transformaciones geométricas se implementa el decorador `prepare_data` que sigue la siguiente estructura:

[6]Un decorador en Python es un tipo de función especial que puede tomar como entrada otra función y modificar su comportamiento.

```
def prepare_data(func, data):
    ...
    preprocessed_data = preprocess(data)
    ...
    transformed_data = func(preprocessed_data)
    ...
    postprocessed_data = postprocess(transformed_data)
    return postprocessed_data
```

El decorador recibe como parámetro la función de transformación y el dato a transformar. Antes de llamar a la función realiza internamente el preprocesado de los datos (concatenar elementos, cargarlos en GPU..), y antes de devolver la salida el post-procesado (restaurar el diccionario). Utilizando esta estructura, basta con implementar cada una de las transformaciones internas (trabajando ya sobre los datos procesados) y añadirle al método la directiva `@prepare_data`.

Análogamente, para las transformaciones de color se construye el decorador `@prepare_data_for_opencv`. En este caso, su trabajo será el de identificar la imagen en el elemento de entrada, transformarlo (si es necesario a `numpy`) y pasar la imagen como parámetro a la función correspondiente . Finalmente, restaura y devuelve el elemento completo (con la imagen transformada).

A cada una de las transformaciones implementadas, se le añade la directiva `@prepare_data_for_opencv` haciendo que estas funciones puedan trabajar directamente sobre imágenes.

## 5.3. *Sprint* 3

### 5.3.1. *Sprint Backlog*

La siguiente épica con mayor prioridad para el proyecto, por ser necesaria para la implementación de las demás, es la construcción del *pipeline* o módulo de composición de operaciones (**FTR.03**). Se decidió dividir este módulo en 2 historias de usuario:

- **HU.03** - *Pipeline*: construcción de un pipeline básico que se encargue de la composición de operaciones.
- **HU.04** - Personalización del *pipeline*: mejora del pipeline básico, permitiendo la elección de diferentes parámetros para su ejecución.

Por otro lado, también se valora la épica visualización de transformaciones (**FTR.02**) dado que su implementación facilita la depuración del propio sistema. Si se pueden visualizar las transformaciones de manera directa, se pueden depurar

más fácilmente los módulos que se vayan implementando. Esta épica también se divide en 2 historias de usuario:

- **HU.05** - Visualización de transformación individual: herramienta de visualización de las transformaciones individuales.
- **HU.06** - Visualización de transformación de lotes: adaptación de la herramienta de visualización de operaciones y elementos de entrada individuales a la visualización de lotes de imágenes para el *pipeline*.

Para este *sprint* se decide realizar las historias de usuario **HU.03** y **HU.05**. Se escoge de esta manera para asegurar que, antes de empezar a refinar el *pipeline*, este ha sido validado por el cliente. De la misma manera, antes de adaptar la herramienta de visualización a estructuras de lotes de imágenes interesa que el cliente valide la versión inicial. Como ambas historias de usuario implican la construcción y diseño de módulos desde cero, se estima que cada una de ellas supone 1 semana de duración.

| *Sprint* Backlog | |
|---|---|
| **Nº de *sprint*** | *Sprint* 3 |
| **Objetivo** | Implementar una primera versión del *pipeline*, es decir, el módulo de composición de operaciones, que admita conjuntos de operaciones y lotes de elementos como entrada. Paralelamente se implementará la herramienta de visualización de transformaciones individuales. |
| **Duración** | 2 semanas. |
| **Items asignados** | 1. **HU.03**: *Pipeline* (1 semana).<br>2. **HU.05:** Visualización transformación individual (1 semana). |

A continuación se detalla el modo de desarrollo de los dos módulos a abordar en este *sprint*.

### 5.3.2. *Pipeline*

Para la implementación de cualquier módulo desde cero es importante analizar sus requisitos.

#### Análisis

Los requisitos identificados para el *pipeline* inicial son:

- El formato de los datos de entrada debe ser el mismo que el del resto de la librería. Es decir, debe trabajar con diccionarios Python con el formato explicado en el apartado anterior.

- Debe admitir lotes de elementos de entrada, es decir, listas de diccionarios Python a los que se aplicarán las transformaciones.

- Todas sus operaciones deben admitir un parámetro de probabilidad del cual dependerá a cuantas imágenes del lote se le aplica cada transformación.

### Diseño

La función principal del pipeline es la composición de operaciones. Para ello, el sistema diferenciará 3 tipos de operaciones:

1. **Operaciones espaciales**: pueden componerse mediante la multiplicación de sus matrices de transformación.

2. **Operaciones de color**: operaciones a nivel de píxel puntuales (el valor del píxel de salida solo depende del píxel de entrada). Pueden expresarse como una función matemática sobre el valor del píxel de entrada.

   Para componerlas, se debe obtener la fórmula matemática propia de cada transformación ($p_t = f(p)$) y se calcula la composición de funciones ($f_c = g(f(x))$. Una vez obtenida la fórmula compuesta, se construye una LUT que contendrá el mapeo directo de cada píxel de entrada con el transformado por el conjunto de operaciones de color.

3. **Operaciones independientes**: operaciones que no admiten composición, como el desenfocado o la ecualización de histograma. Estas operaciones se aplican de manera individual a la imagen de entrada.

El diseño del módulo se muestra en la Figura 5.6. Al combinar en la API código funcional y código orientado a objetos, para visualizar el funcionamiento se incluyen en el diagrama de clases los módulos que se utilizan en gris. El objeto central es el `Pipeline`, encargado de la ejecución y transformación de cada lote de elementos. Por cada una de las operaciones que puede integrar, se construye una clase `Pipeline Operation`, del subtipo que corresponda. Estas clases, reutilizarán el código ya implementado para las transformaciones independientes de los módulos `geometric_ops_functional` y `pixel_ops_functional` (código implementado para las operaciones independientes). Por último, el módulo `pipeline_functional` incluye todas aquellas operaciones de cierto nivel de complejidad empleadas por el `Pipeline`, como el preprocesado de los datos o la composición de funciones.

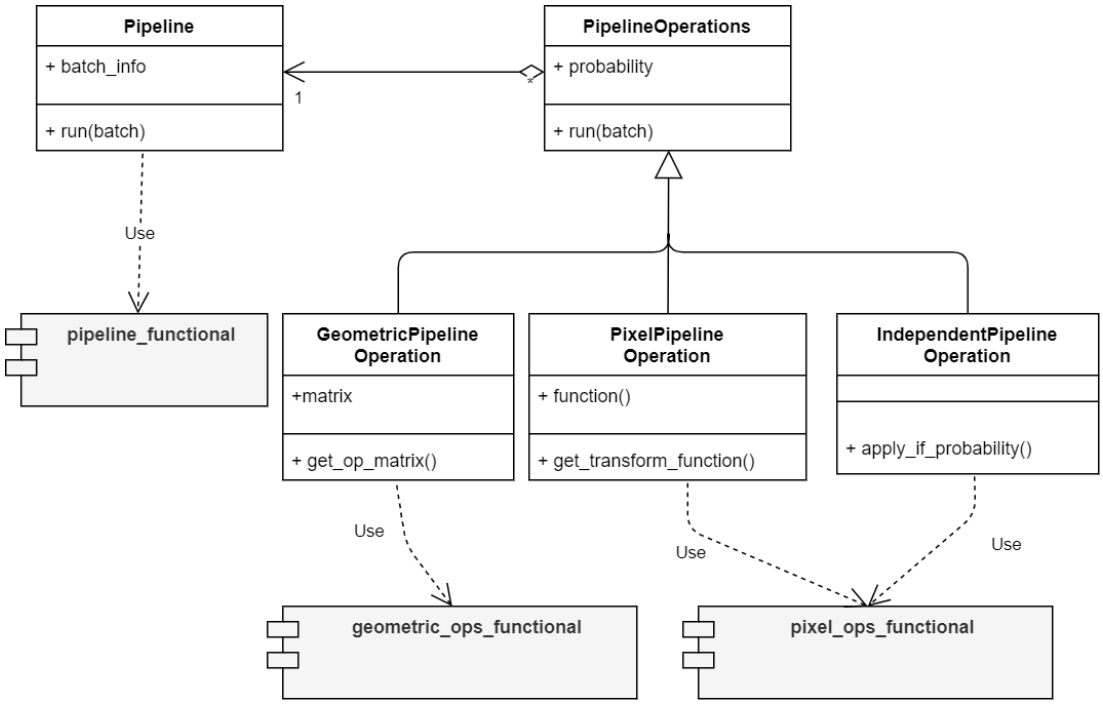



Figura 5.6: Diagrama de clases del módulo *pipeline*, mostrando en gris los módulos que utiliza cada clase.

El funcionamiento del *pipeline* se ilustra en la Figura 5.7. Los pasos de ejecución por cada uno de los elementos del lote de entrada son los siguientes:

1. Divide las operaciones de entrada en las 3 clases correspondientes y calcula cuales debe aplicar a este elemento en base a su probabilidad.

2. Construye la LUT con la composición de transformaciones de color.

3. Identifica la imagen en el elemento de entrada (si hay) y calcula la imagen transformada aplicando la LUT.

4. Aplica, si hay, las operaciones independientes sobre la imagen.

5. Una vez transformada la imagen, hace un preprocesado del diccionario de entrada idéntico al de las transformaciones geométricas (generando un tensor compuesto por imagen y mapas de calor, un segundo tensor de máscaras y mapas de segmentación y la matriz de coordenadas homogéneas).

6. Calcula la matriz de transformación correspondiente a la composición de operaciones geométricas.

7. Aplica la transformación afín correspondiente a esta matriz a los 2 tensores de datos. A la matriz de coordenadas de puntos se le aplica la multiplicación matricial directamente.

8. Finalmente hace el post-procesado de datos (idéntico al de las transformaciones geométricas) restaurando el formato de diccionario de entrada. Por defecto la salida estará como `tensor torch` en GPU ya que así puede servir de entrada directa para una red neuronal Pytorch.

### Implementación

La implementación del *pipeline* siguió el esquema mostrado en el diagrama de la Figura 5.6. Un ejemplo de inicialización y ejecución del pipeline sería:

```
pipeline = Pipeline(pipeline_operations=(
                ScalePipeline(probability=0.5, scale_factor=0.5),
                ShearPipeline(probability=0.3, shear=(0.2, 0.2)),
                TranslatePipeline(probability=0.4,
                                  translation=(10,50)),
                HflipPipeline(probability=0.6)
            ))

transformed_batch = pipeline(batch)
```

Cabe destacar que para muchas de las operaciones del *pipeline* es necesaria cierta información sobre los datos a tratar (como el ancho de la imagen para el *flip* horizontal). Como el *pipeline* debe ser sencillo de usar, en lugar de obligar al usuario a ingresar este tipo de parámetros, es el *pipeline* quien se encarga de extraerlos de los elementos de entrada. Se analiza el primer elemento de entrada que transforma el pipeline y se extrae toda la información necesaria para su correcto funcionamiento. Esta información se almacena para todos los elementos futuros que procese el *pipeline*.

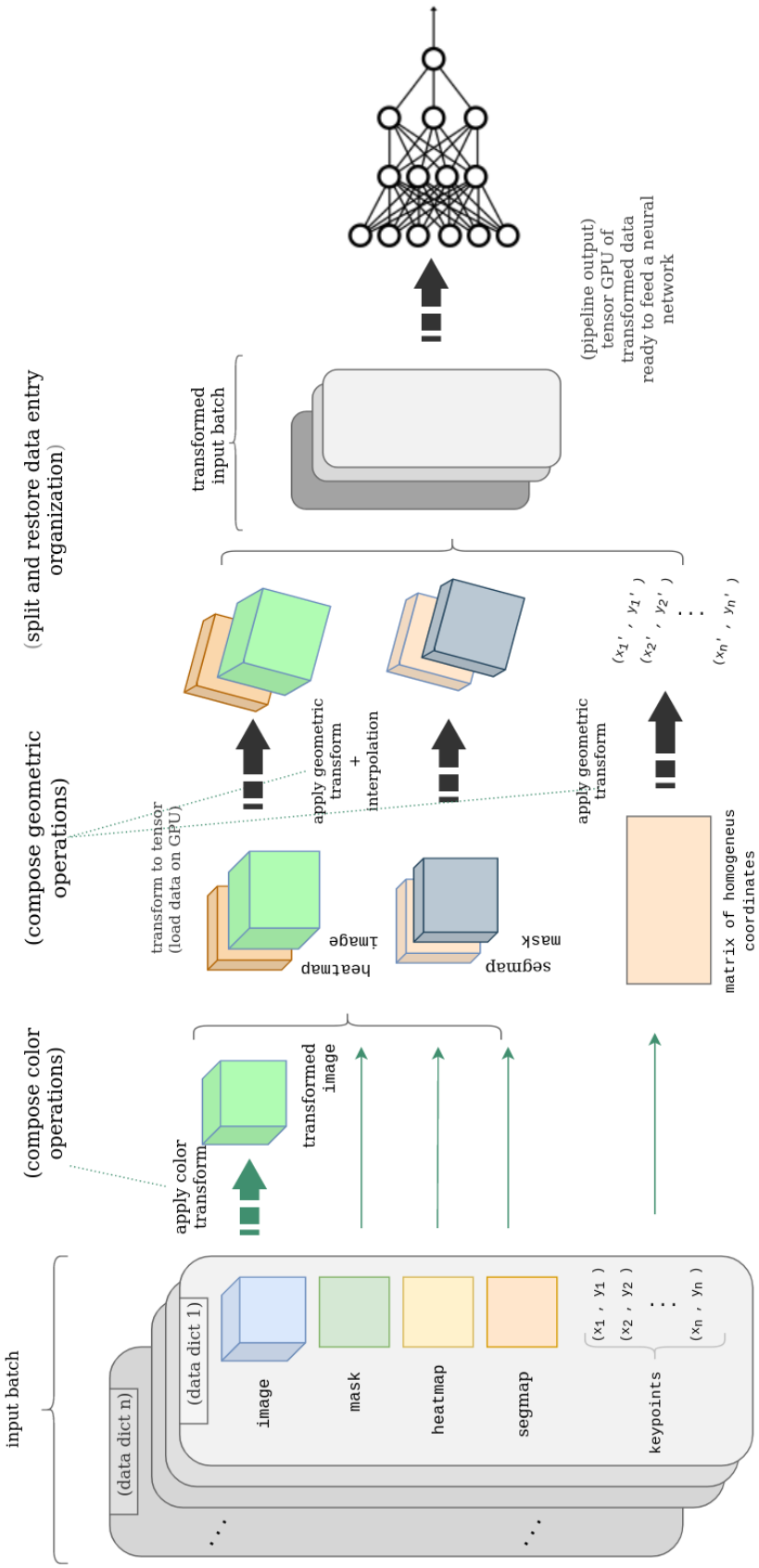



Figura 5.7: Esquema de módulo de composición de operaciones (*pipeline*).

### 5.3.3. Herramienta de visualización

De nuevo, al suponer la construcción completa de este módulo, cobra gran importancia la parte de análisis y diseño.

#### Análisis

Los requisitos identificados para la herramienta de visualización son:

- Debe mostrar tanto el elemento original como el elemento transformado, para poder comparar de forma sencilla el impacto de la transformación.
- Debe poder seleccionarse de manera interactiva que elementos, o combinación de elementos, se muestran en cada momento.
- Si el elemento de entrada contiene elementos no bidimensionales, como pueden ser etiquetas, también deben poder visualizarse para dar una visión completa de los elementos de entrada.

#### Diseño

La herramienta de diseño debe ser sencilla y clara para una visualización directa de los elementos. Se diseñó siguiendo el esquema de la Figura 5.8. Todos los elementos que compongan el dato transformado se superpondrán en la figura correspondiente. A la derecha existirá un menú con una entrada por cada elemento que compone el dato, donde se puede seleccionar si se visualiza o no. Al final de este menú se incluirá de manera textual los datos no bidimensionales del elemento.

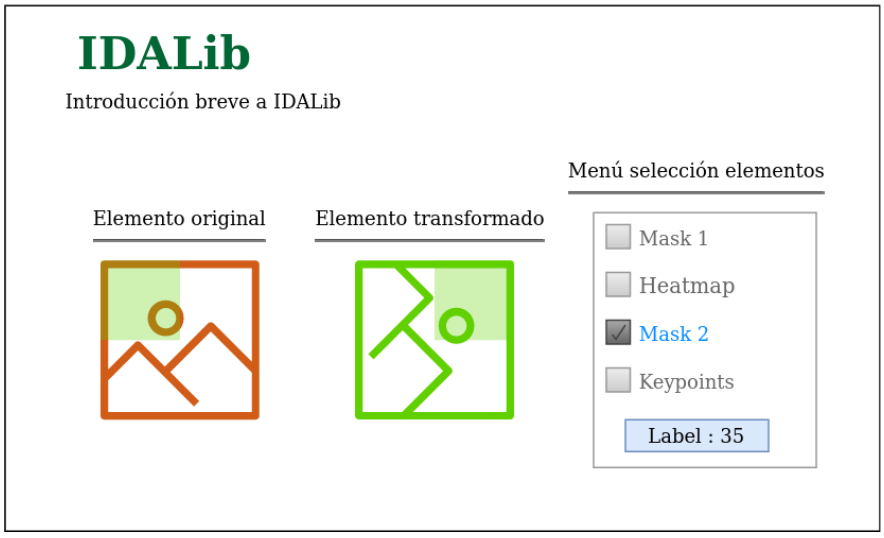



Figura 5.8: Esquema de diseño de elementos de la interfaz.

**Implementación**

Para la implementación, se incorpora un nuevo sub-módulo de visualización, que genera un "documento" HTML con Bokeh. Para su invocación, se añade el parámetro `visualize` a cada una de las transformaciones individuales. La llamada a la rutina de visualización se hace a través de los decoradores ya explicados (tanto el de transformaciones espaciales como el de transformaciones a nivel de píxel) por ser común a todas las transformaciones.

El aspecto de la herramienta se muestra en la Figura 5.9, en la que se puede ver un ejemplo de elemento con 2 máscaras (generadas automáticamente) y 4 puntos. Las máscaras, mapas de segmentación y mapas de calor se superponen sobre la imagen con una opacidad reducida para poder ver su correspondencia con las secciones de la imagen.

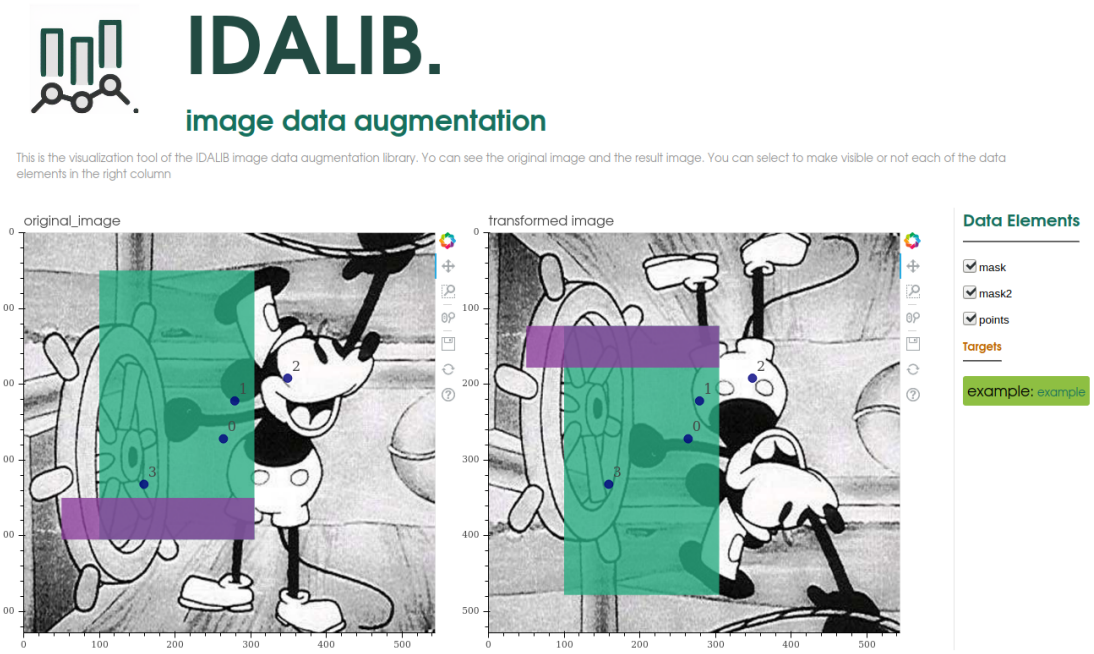



Figura 5.9: Aspecto de la interfaz de visualización de IDALib.

## 5.4. *Sprint* 4

Una vez validados por el cliente tanto el *pipeline* básico como la herramienta de visualización, se decide continuar y terminar estas dos herramientas.

### 5.4.1. *Sprint Backlog*

En este cuarto *sprint* se continúa con el trabajo sobre el *pipeline* y sobre la herramienta de visualización dado que el cliente ya dio el visto bueno a la versión

inicial de ambas. Por lo tanto a este *sprint* se le asignan las siguientes historias de usuario:

- **HU.04** - Personalización del *pipeline*.
- **HU.06** - Visualización de la transformación del lote.

Ambas historias de usuario, al ser la continuación del trabajo del *sprint* anterior, se estiman en una duración de 0,5 semanas cada una.

| *Sprint* Backlog | |
|---|---|
| **Nº de *sprint*** | *Sprint* 4. |
| **Objetivo** | Completar la implementación del *pipeline* para que acepte múltiples parámetros de personalización para su funcionamiento. Adaptar la herramienta de visualización de las transformaciones individuales para que acepte lotes de imágenes y poder integrarlo con el *pipeline*. |
| **Duración** | 1 semana. |
| **Items asignados** | 1. **HU.04**: Personalización del *pipeline* (0,5 semanas).<br>2. **HU.06**: Visualización transformación de lote (0,5 semanas). |

### 5.4.2. Personalización del *pipeline*

En esta sección se detallan las fases llevadas a cabo para incluir la personalización del *pipeline* en el sistema.

#### Análisis

Los requisitos identificados a añadir para el *pipeline* son:

1. Integrar operaciones con comportamiento aleatorio. Por ejemplo, para una rotación, indicar un intervalo y que cada transformación seleccione un valor aleatorio dentro del mismo. De las operaciones ya implementadas, las que admiten un comportamiento de este tipo son:
   - Rotación.
   - Escalado.
   - *Shear*.
   - Cambio de brillo.

   - Cambio de contraste.
   - Corrección gamma.

2. Selección del modo de interpolación para las transformaciones espaciales (vecino más cercano o bilinear).

3. Selección del modo de extrapolación (ceros, reflexión o borde).

4. Selección del formato de salida: como es común que la entrada a una red neuronal sea directamente una tupla con cada uno de los elementos que componen el dato, debe poder seleccionarse el formato de salida del pipeline (diccionario o tupla).

5. Seleccionar el tipo de dato de salida (profundidad de color), por defecto se mantiene el tipo de dato de entrada, pero es posible que para la entrada de una red neuronal interese un tipo de dato concreto.

6. Permitir redimensionar todos los elementos de entrada a un tamaño dado. La entrada a las redes neuronales por lo general debe ser lo más homogénea posible y, en lo que respecta a las dimensiones de la imagen, interesa que el *pipeline* se encargue de esta transformación.

7. Permitir indicar un mapeo de puntos en las operaciones de *flip* horizontal y vertical. Como se puede ver en la imagen de la Figura 5.10, el primer punto marca el borde izquierdo de la mandíbula y el segundo el borde derecho. Al hacer el *flip* horizontal, la imagen generada no es consistente (ya que la red estaría asumiendo que el punto izquierdo está en el nuevo lado derecho de la imagen y viceversa). Por ello es importante permitir modificar el orden de los puntos tras la operación de *flip*. En el ejemplo la imagen se intercambiarían las coordenadas de los puntos 1 y 2 para que el resultado sea consistente.

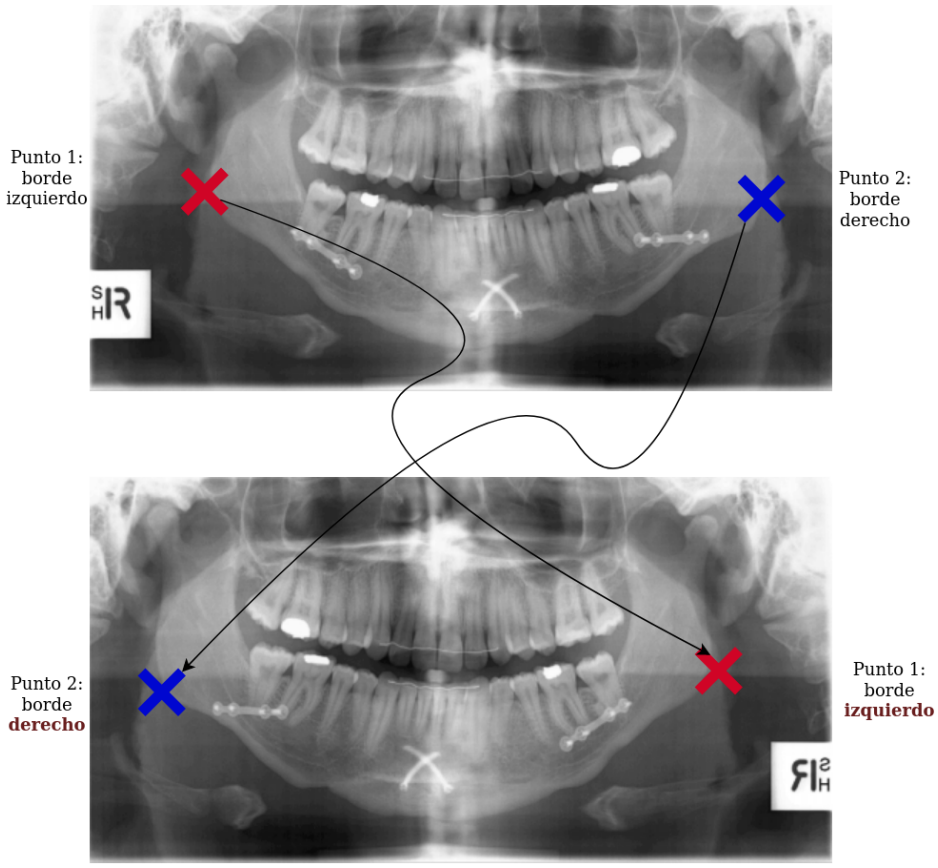



Figura 5.10: Error de etiquetado de los puntos característicos después de la operación de *flip* horizontal.

El diseño del *pipeline* y de sus sub-módulos continúa siendo el mismo que el especificado en el *sprint* anterior. Respecto las operaciones aleatorias añadidas, se especifican en el Cuadro 5.6 (operaciones geométricas) y en el Cuadro 5.5 (operaciones a nivel de píxel).

Cuadro 5.5: Especificación de operaciones a nivel de píxel aleatorias y sus parámetros

| Operación | Parámetros |
|---|---|
| **RandomBrightness** | ▪ **brightness_range**: rango de factor de brillo. |
| **RandomContrasts** | ▪ **contrast_range**: rango de factor de contraste. |
| **RandomGamma** | ▪ **gamma_range**: rango de factor gamma. |

Cuadro 5.6: Especificación de operaciones geométricas aleatorias y sus parámetros.

| Operación | Parámetros |
|---|---|
| **RandomScale** | ▪ **scale_range**: rango de factor de escalado. Se utiliza un mismo rango para el escalado en ambos ejes.<br>▪ **keep_aspect**: si es True, el valor de escalado en ambos ejes será el mismo, de lo contrario puede ser diferente.<br>▪ **center_deviation**: cantidad de píxeles máximo que se puede desviar el centro de escalado del centro original. Con este parámetro se consigue una variabilidad extra: además de cambiar el rango de escalado, el centro de escalado varía ligeramente creando diferentes resultados.<br>▪ **center**: centro de rotación original. |
| **RandomRotate** | ▪ **degrees_range**: rango de grados de rotación.<br>▪ **center_deviation**: número máximo de píxeles que se puede desviar el centro de rotación del centro original. Con este parámetro se consigue una variabilidad extra: además de cambiar el rango de rotación, el centro de rotación varía ligeramente<br>▪ **center**: centro de rotación original. |
| **RandomTranslate** | ▪ **translation_range:** rango de píxeles de traslación. Se utiliza un mismo rango para la traslación en ambos ejes.<br>▪ **same_translation_on_axis**: indica si la traslación en ambos ejes debe ser igual o puede ser diferente. |
| **RandomShear** | ▪ **shear_range**: rango de píxeles de shear. Se utiliza un mismo rango para el shear en ambos ejes.<br>▪ **same_shear_on_axis**: indica si la cantidad de shear en ambos ejes debe ser igual o puede ser diferente. |

En relación a los nuevos parámetros para el *pipeline* cabe mencionar que, en el caso de seleccionar `nearest` como modo de interpolación, todos los elementos bidimensionales se pueden concatenar en un mismo elemento (ya que se separaban las máscaras y mapas de segmentación en un elemento separado para aplicarles este tipo de interpolación). Se modifica el preprocesado de datos de manera que

si la interpolación es `bilineal`, mantiene el comportamiento ya definido, pero si es `nearest` se concatenan en un único elemento.

Los parámetros de personalización del pipeline se pasarán como argumento en su inicialización, quedando las opciones indicadas en el Cuadro 5.7. La mayor ventaja de emplear valores por defecto es que se ofrece la flexibilidad de poder inicializar el pipeline de la manera más sencilla:

```
pip=Pipeline(pipeline_operations=(
        ScalePipeline(probability=0.5, scale_factor=0.5),
        ShearPipeline(probability=0.3, shear=(0.2, 0.2)),
        HflipPipeline(probability=0.6)
        ))
```

Hasta poder realizar la inicialización más personalizada con todos los detalles deseados:

```
pip=Pipeline(
       resize=(200,500),
       interpolation='nearest',
       padding_mode='reflexion',
       output_format='tuple',
       pipeline_operations=(
          ScalePipeline(probability=0.5, scale_factor=0.5),
          ShearPipeline(probability=0.3, shear=(0.2, 0.2)),
          TranslatePipeline(probability=0.4,
                            translation=(10,50)),
          HflipPipeline(probability=0.6,
                        exchange_points=[(0, 5), (1, 6)]),
          RandomRotatePipeline(probability=0.4,
                               degrees_range=(-20, 20),
                               center_deviation=15),
          RandomScalePipeline(probability=1,
                              scale_range=(0.5, 1.5))
          ))
```

Cuadro 5.7: Especificación de parámetros del *pipeline*.

| Parámetro | Descripción | Valor por defecto |
|---|---|---|
| `pipeline_operations` | Lista de objetos pipelineOperations inicializado. Indica las transformaciones a implementar por el pipeline. | – |
| `resize` | Tupla del tamaño deseado para los elementos de entrada. Si no se indica, no se aplica el redimensionado. | None |
| `interpolation` | Modo de interpolación para las operaciones geométricas ('nearest' o 'bilinear'). | 'bilinear' |
| `padding_mode` | Modo de extrapolación para las operaciones geométricas ('zeros', 'reflexion' o 'border'). | 'zeros' |
| `output_format` | Formato de los elementos de salida del pipeline ('tuple' o 'dict') | 'dict' |
| `output_type` | Tipo de dato de salida (uint8, uint32, float64...) . Si no se indica, se mantiene el tipo de dato de entrada. | None |

### 5.4.3. Visualización de la transformación de un lote

#### Análisis

En este caso, la herramienta de visualización de transformaciones se debe adaptar a la visualización de lotes de elementos, para permitir la visualización de las transformaciones realizadas por el *pipeline*.

Para ello, es importante poder seleccionar en la interfaz el elemento a visualizar. No es necesario poder visualizar todos y cada uno de los elementos del lote, basta con una muestra representativa del lote y de las transformaciones llevadas a cabo.

En la Figura 5.11 se ilustra el diseño de esta interfaz, que es prácticamente idéntico al anterior diseño. Simplemente se le añade un elemento superior que permita seleccionar el *item* a visualizar.

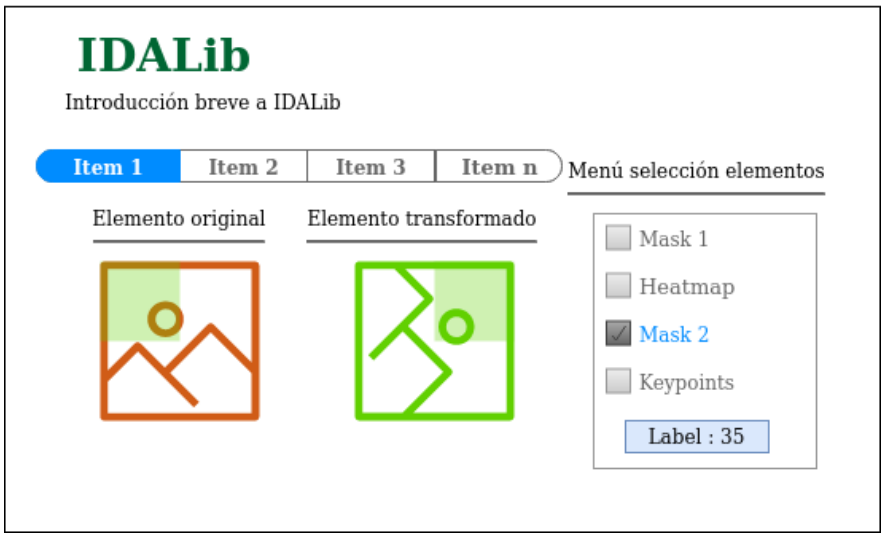



Figura 5.11: Diseño de la interfaz de visualización de lotes.

### Implementación

Para la implementación se emplea la estructura de datos de Bokeh **tab**, que permite crear varias ventanas y elegir cual visualizar de manera interactiva. La generación de cada una de las ventanas se realizó con el mismo código de las transformaciones individuales para evitar duplicados. Para su ejecución, se añade el parámetro `visualize` en la llamada a ejecución del *pipeline*.

El número de elementos a visualizar se puede modificar, pero se limita por defecto a 5 elementos para evitar sobrecargas en el navegador. El aspecto final es el mostrado en la Figura 5.12: se muestra una prueba con un lote compuesto por elementos idénticos para visualizar la variabilidad en las transformaciones.

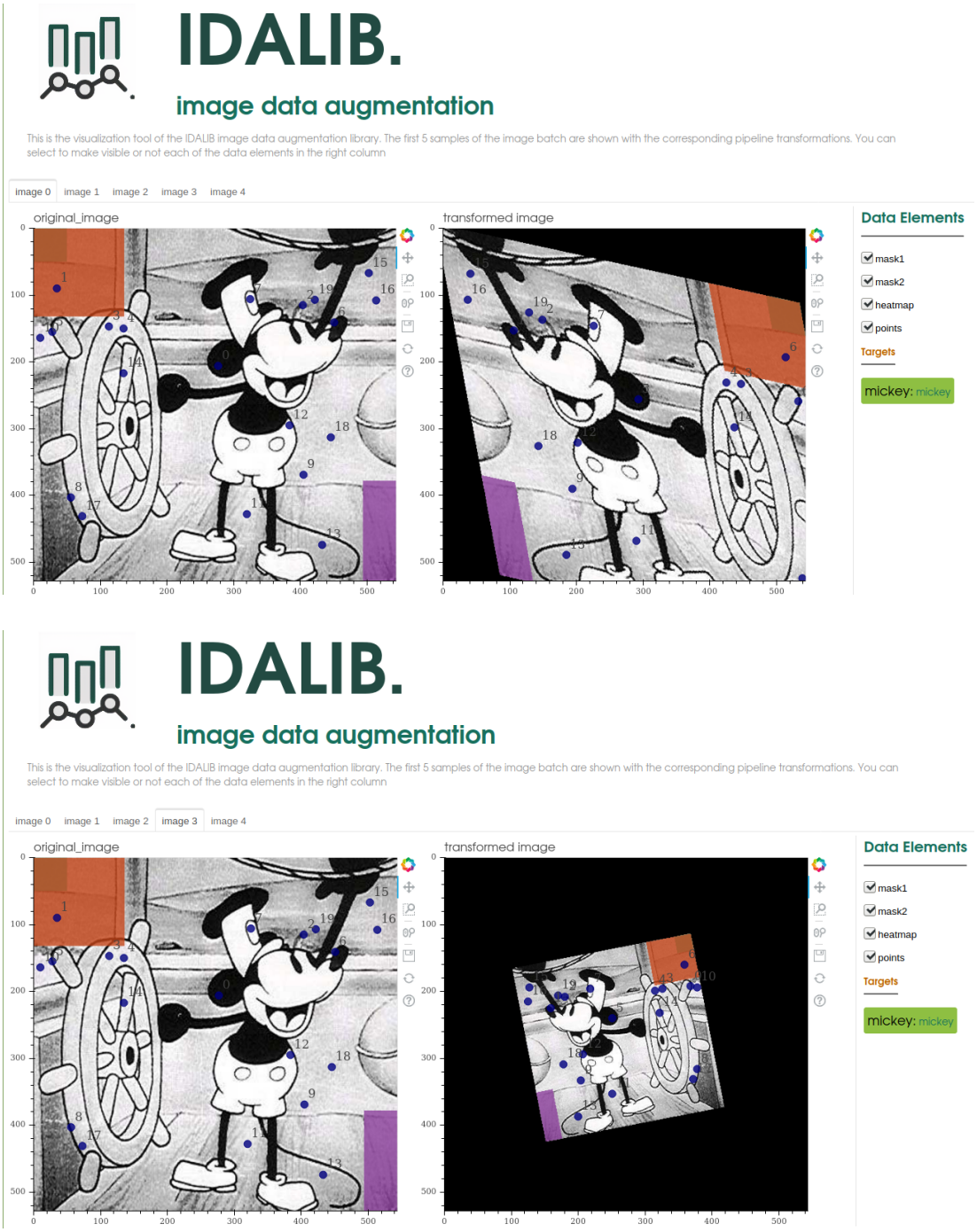



Figura 5.12: Implementación de la interfaz de visualización por lotes. Ejemplo de visualización de transformación de un lote de imágenes idénticas. La primera imagen representa la transformación realizada sobre el primer elemento del lote. En la segunda se visualiza la transformación aplicada al cuarto elemento del lote.

## 5.5. *Sprint* 5

En este momento del proyecto, de las 6 épicas iniciales ya se han completado 3 (Operaciones de transformación, Visualizar transformaciones y *Pipeline*), quedando pendientes otras tres:

- **FTR.04** - *Dataloader*.
- **FTR.05** - *Image data augmentation* a disco.
- **FTR.06** - Publicación y estandarización.

La publicación y estandarización de la librería implican necesariamente tener completada la implementación, por ello se aplaza para *sprints* futuros.

### 5.5.1. *Sprint Backlog*

Tanto la construcción del módulo de aumento a disco como el *dataloader* se estiman en una duración de 1 semana cada una. Estas épicas no se subdividen, quedando ambas como historias de usuario.

Al ser ambos módulos de alto nivel, interesa que tengan una interfaz homogénea respecto a los parámetros de entrada, por lo que se asignan ambas historias de usuario a este *sprint* para poder realizarlas en paralelo.

| *Sprint* Backlog | |
|---|---|
| **Nº de *sprint*** | *Sprint* 5 |
| **Objetivo** | Construcción del módulo *dataloader* que se encargue de hacer directamente el *image data augmentation* y sirva como entrada a una red neuronal. Construcción del módulo que implemente el *image data augmentation* sobre disco. Ambas herramientas se construyen sobre el objeto ya implementado de *pipeline*. |
| **Duración** | 2 semanas. |
| **Items asignados** | 1. **HU.07:** *Dataloader* (1 semana).<br>2. **HU.08**: Implementación del *image data augmentation* a disco (1 semana). |

### 5.5.2. *Dataloader*

#### Análisis

En lo que refiere al módulo de entrada para la red neuronal se identifican los siguientes requisitos:

- Debe poder definirse el tamaño de lote y ser un objeto iterable para servir como entrada a la red neuronal.
- Debe permitir recorrer más de una vez el mismo *dataset*, generando un orden de datos diferente y transformaciones distintas en cada iteración. Esto lo hace compatible con las épocas de entrenamiento de una red neuronal (deben seguir un orden diferente y aplicar diferentes transformaciones para evitar el sobreajuste[7]).
- Debe ser una herramienta flexible e intuitiva para el usuario.
- Debe admitir todas las operaciones y opciones de personalización del *pipeline*.

Para la implementación del módulo, en vez de realizar una implementación desde cero, se valoran las 2 estructuras más comunes del sector: el *dataloader* y el generador. En el Cuadro 5.8 se muestran sus diferencias más relevantes.

Cuadro 5.8: Principales diferencias entre *dataloader* y generadores.

| ***Dataloader*** | **Generador** |
|---|---|
| ▪ Propio de Pytorch.<br>▪ Implica la construcción del método `get_item`, lo que hace que trabaje sobre elementos individuales.<br>▪ Trabaja con objetos *dataset* de Pytorch, haciendo más sencilla su implementación. | ▪ General de Python.<br>▪ Puede trabajar directamente sobre conjuntos de elementos.<br>▪ No utiliza subclases externas, haciéndolo más flexible pero implicando mayor requerimiento de implementación. |

De ambas opciones se selecciona el *dataloader*, puesto que sus principales limitación de trabajar sobre elementos individuales no supone un problema ya que el *pipeline* de IDALib trabaja así.

[7]Efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado.

### Diseño

El diseño para el módulo es el mostrado en el diagrama de la Figura 5.13. Para este diseño se tuvieron en cuenta los siguientes aspectos:

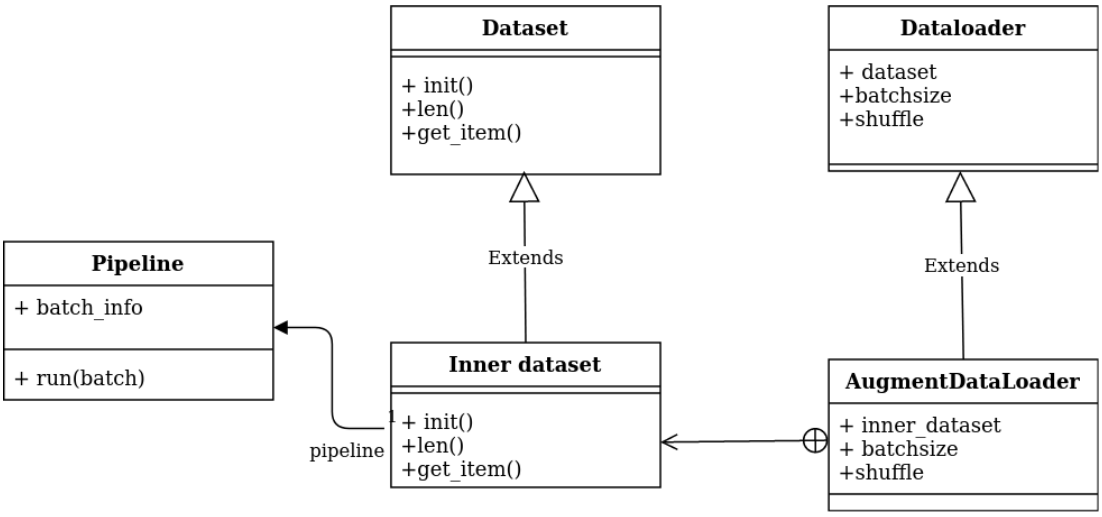



Figura 5.13: Diagrama de clases del módulo *dataloader*.

- El objeto *dataloader* de Pytorch trabaja con su tipo de dato propio: *Dataset*. El *Dataset* es la clase encargada de la lectura/creación de los datos que forman parte del *dataset* de entrada. Tiene 3 métodos principales:

  - `init`: inicialización.
  - `len`: devuelve el número de elementos del *dataset*.
  - `get_item(index)`: devuelve el elemento de la posición *index* del *dataset*.

  El *dataloader* es el encargado de gestionar este conjunto de datos. Es un objeto iterable que gestiona el orden de los elementos, los lotes de elementos...

- Para realizar el *image data augmentation* es necesario que cada elemento que devuelva el objeto *dataset* pase por nuestro *pipeline*. Para esto, se construye una sub-clase del objeto *dataset* que incorpore el objeto *pipeline* (`Inner dataset`) sobreescribiendo el método `get_item`.

- Como la idea es tener un objeto compacto y sencillo para el usuario, este (`Inner dataset`) se incluye como una clase interna a la clase `AugmentDataloader`.

- Se construye la clase `AugmentDataloader` que es la que el usuario empleará e instanciará. Ésta hereda de *dataloader*, para incorporar las funcionalidades

del mismo pero utilizando como *dataset* interno una instancia del objeto `Inner Dataset`.

- De esta forma conseguimos que con un único objeto `AugmentDataloader`, el usuario solo tenga que indicar los parámetros deseados para su gestión sin necesidad de crear clases adicionales ni pasos intermedios.

### 5.5.3. Implementación

La implementación del `AugmentDataloader` admite los parámetros de inicialización mostrados en el Cuadro 5.9.

Cuadro 5.9: Especificación de parámetros de inicialización del `AugmentDataloader`.

| Parámetro | Descripción | Valor por defecto |
|---|---|---|
| `batchsize` | Indica el número de elementos por cada lote. | 1 |
| `dataset` | *Dataset* con el que se construye el `innerdataset` | – |
| `shuffle` | Indica si el orden de los elementos debe ser el del *dataset* o debe seguir un orden aleatorio. | True |
| `pipeline_operations` | Lista de pipelineOperations para el pipeline | — |
| `resize` | Tamaño deseado en caso de redimensionado de los elementos. | None |
| `interpolation` | Modo de interpolación de las transformaciones geométricas para el pipeline. | 'bilinear' |
| `padding_mode` | Modo de extrapolación (ceros, reflexión o borde) | 'zeros' |
| `output_format` | Formato de los elementos de salida (dict o tuple) | 'dict' |
| `output_type` | Tipo de dato de salida. | None |

### 5.5.4. Módulo de *image data augmentation* a disco

#### Análisis

Se identifican los siguientes requisitos para la construcción del módulo de *image data augmentation a disco*:

- El usuario debe poder elegir el número de elementos a generar por cada elemento de entrada. Además, cada elemento generado a partir de un mismo elemento de entrada debe ser diferente.
- El módulo debe almacenar diferentes tipos de datos en memoria (máscaras, mapas de calor, coordenadas de puntos..) de manera que se entienda qué tipo de dato son y a qué elemento de entrada corresponden. Por ejemplo, un elemento `item_1` formado por 1 máscara y un mapa de calor, debe poder almacenarse de manera que se entienda qué elemento es la máscara, cual es el mapa de calor y que ambos se asocian al elemento `item_1`.
- Existen formas muy variadas para almacenar los *datasets* en disco (organización por directorios, por archivos csv...). Debe ser una herramienta flexible que permita al usuario almacenar los elementos de la forma que desee.

#### Diseño

El diseño del módulo es el ilustrado en el diagrama de la Figura 5.14. Básicamente, se compone de una única clase (`AugmentToDisk`) que usa el *pipeline*. Esta clase trabaja con un *dataset* como elemento de entrada, ya que interesa usar los mismos tipos de datos de entrada que en el `AugmentDataloader` para ofrecer una interfaz de parámetros uniforme en todos los módulos la librería.

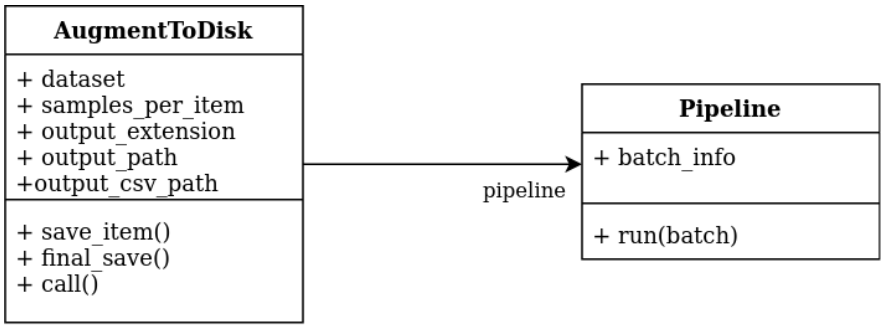



Figura 5.14: Diagrama de clases del módulo de *image data augmentation* a disco.

El modo de funcionamiento del objeto se ilustra en la Figura 5.15. El sistema trabaja con 2 parámetros principales: el *dataset* de entrada, y el número de elementos a generar por cada elemento del *dataset*. El funcionamiento del sistema es el siguiente:

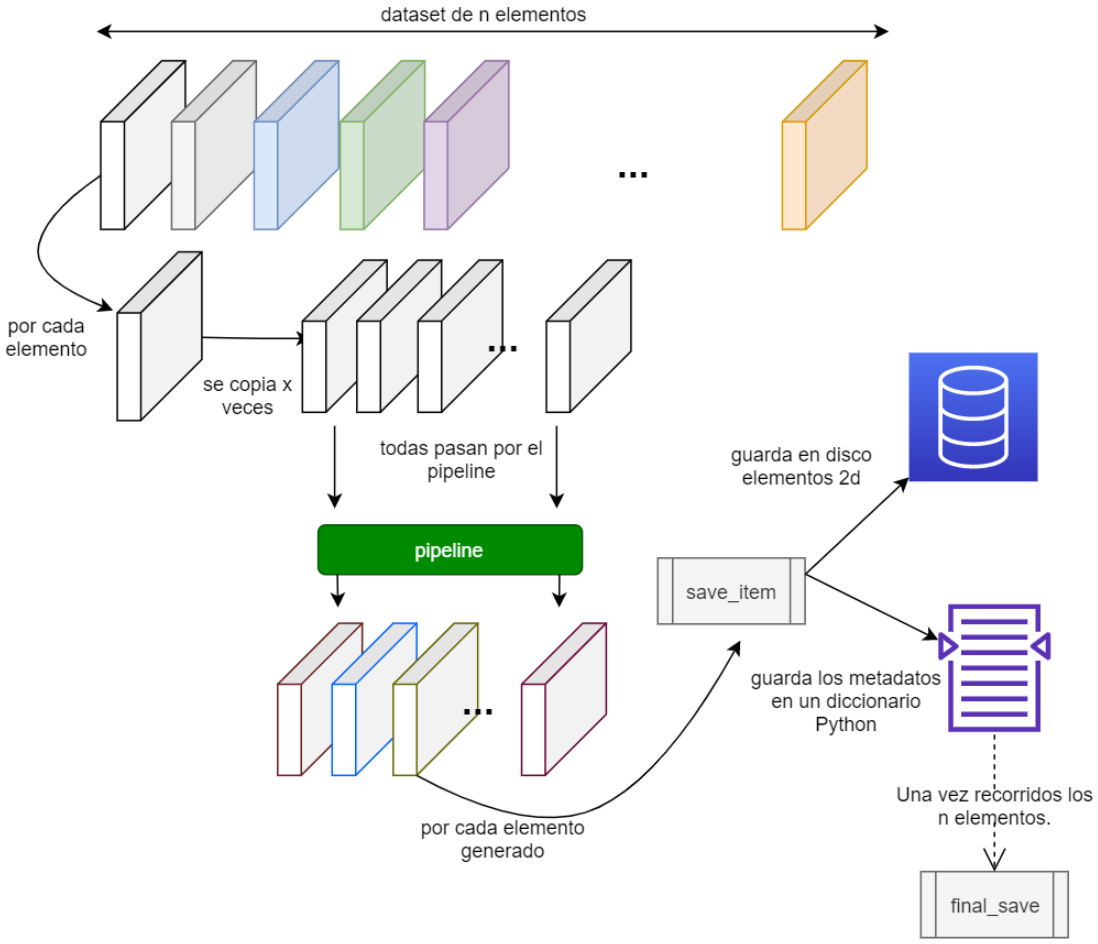



Figura 5.15: Esquema de funcionamiento del módulo de *image data augmentation* a disco.

1. Por cada elemento del *dataset* de entrada:
   a) Se copia $x$ veces (siendo $x$ el número de elementos a generar por cada elemento de entrada)
   b) Este conjunto de $x$ copias, se hace atravesar el pipeline.
   c) Por cada uno de los $x$ elementos ya transformados, se llama al método `save_item`. Éste se encarga de guardar los datos bidimensionales a disco y, los metadatos como las coordenadas de puntos o etiquetas los almacena en un diccionario interno.
   d) Una vez hecho esto las $x$ veces, se sigue con el siguiente elemento del *dataset* hasta recorrerlo entero.
2. Una vez recorrido todo el *dataset* se ejecuta el método `final_save`. Este método se encarga de generar un archivo csv con los metadatos que se almacenaron en el diccionario interno y guardarlo en disco.

### Implementación

La implementación final del *AugmentToDisk* admite los parámetros de inicialización mostrados en el Cuadro 5.10.

Cuadro 5.10: Especificación de parámetros de inicialización del `AugmentToDisk`.

| Parámetro | Descripción | Valor por defecto |
|---|---|---|
| `dataset` | *Dataset* sobre el que se realiza el *image data augmentation*. | – |
| `samples_per_item` | Número de elementos generados por cada elemento del *dataset* de entrada. | 2 |
| `total_output_samples` | Sustituye a `samples_per_item` y calcula de manera interna el número de elementos a generar sobre el *dataset* de entrada para alcanzar el valor indicado. | None |
| `pipeline_operations` | Lista de `pipelineOperations` para el *pipeline*. | — |
| `resize` | Tamaño deseado en caso de redimensionado de los elementos. | None |
| `interpolation` | Modo de interpolación de las transformaciones geométricas para el pipeline. | 'bilinear' |
| `padding_mode` | Modo de extrapolación (ceros, reflexión o borde). | 'zeros' |
| `output_type` | Tipo de dato de salida. | None |

Como se mencionó en los requisitos, existen muchas opciones para almacenar un *dataset* en disco. Para ofrecer un buen equilibrio entre facilidad de uso y flexibilidad, se ofrece un modo de guardado por defecto pero también se permite que los usuarios puedan sobrescribirlo para adaptarlo a sus necesidades. El modo por defecto sigue el siguiente formato:

- Cada elemento del *dataset* de entrada se asocia a un id (como puede ser el nombre del archivo leído o simplemente la posición que ocupa en el *dataset*).
- Para guardar los elementos bidimensionales se almacenan todos en el directorio indicado en el parámetro `output_path` (por defecto ./augmented) siguiendo la siguiente convención de nombres:
  - Para las imágenes:
    `<id_image>_<sample_number><extension>`

- Para los demás elementos:
  `<id_image>_<sample_number>-<data_type><extension>`

donde <`sample_number`> hace referencia al número de copia del dato original dado y <`data_type`> al nombre asociado al elemento en el diccionario. Por ejemplo, dado el elemento de entrada del formato: {`'id':1234, 'image':img, 'mask':mask, 'mask2':mask2`} y si se generan 5 elementos por cada elemento de entrada, para almacenar el cuarto elemento generado se crean los archivos:

- `1234_4.jpg` (imagen).
- `1234_4-mask1.jpg` (máscara).
- `1234_4-mask2.jpg` (máscara).

- Para guardar los metadatos, se genera un archivo csv donde cada fila se corresponde a uno de los elementos generados. En la primera columna del csv se disponen 'ids' de cada elemento generado (en el formato `id_0`, `id_1`, `id_2`,...) y en las demás columnas cada tipo de dato a guardar. En el caso de las coordenadas de puntos, por cada uno de los puntos se generan 2 columnas del tipo `(point_0_x, point_0_y)` generando $2 \times n$ columnas (donde $n$ es el número de puntos).

Si el usuario requiere un modo de guardado personalizado, debe crear una subclase a `AugmentToDisk` y sobrescribir los métodos `save_item` y `final_save` con el formato deseada.

## 5.6. Sprint 6

En esta sección se detallan las tareas realizadas en el último *sprint* de desarrollo del proyecto. Analizando las épicas del proyecto, solo falta por completar la épica **FTR.06** - Publicación y estandarización.

### 5.6.1. *Sprint Backlog*

Para alcanzar el objetivo de la épica, ésta se divide en 3 historias de usuario:

- **HU.09** - Estandarización del código (PEP8).
- **HU.10** - Generación de la documentación de la librería.
- **HU.11** - Publicación de la librería y documentación.

En este caso, las 3 historias de usuario se estiman en una duración de menos de 0,5 semanas cada una. Al poder trabajar en paralelo en las 3, se estima que se pueden llevar a cabo en una semana en total. Las 3 se asignan a este último *sprint*.

| *Sprint* Backlog | |
|---|---|
| **Nº de *sprint*** | *Sprint* 6 |
| **Objetivo** | Estandarización y documentación de la librería. Se comprobarán todas las buenas prácticas de la guía PEP8 y se generará la documentación de la librería para finalmente hacer ambas públicas y accesibles para cualquier usuario. |
| **Duración** | 1 semana |
| **Items asignados** | 1. **HU.09** - Estandarización del código (PEP8) (< 0,5 semanas).<br>2. **HU.10** - Generación de la documentación de la librería (< 0,5 semanas).<br>3. **HU.11** - Publicación de la librería y documentación (< 0,5 semanas). |

### 5.6.2. Estandarización del código

Para llevar a cabo la estandarización del código se empleó la herramienta de Pycharm `inspect code`, que identifica todos los errores e incumplimientos de las buenas prácticas. Los *dosctrings*[8] de documentación del código se construyeron siguiendo el formato ***reStructuredText Docstring Format*** (uno de los formatos estándar más extendidos para Python).

### 5.6.3. Generación de la documentación de la librería

Para la generación de la documentación de la librería se empleó la herramienta **Sphinx**. Ésta permite generar la documentación tanto en formato HTML como en PDF automáticamente a partir de los *docstrings* de la API. En las Figuras 5.16 y 5.16 se puede ver el aspecto final de la documentación en formato HTML generada. La documentación incluye las siguientes secciones:

- Introducción a IDALib.
- Primeros pasos (cómo utilizar las funcionalidades básicas de la API).

[8]Las cadenas de documentación en Python (o docstrings) proporcionan una forma conveniente de asociar la documentación con los módulos, funciones, clases y métodos de Python.

- Ejemplos de programas con las diferentes funcionalidades.
- La documentación de la API con la descripción de las funciones y parámetros incluidos (Figura 5.17).

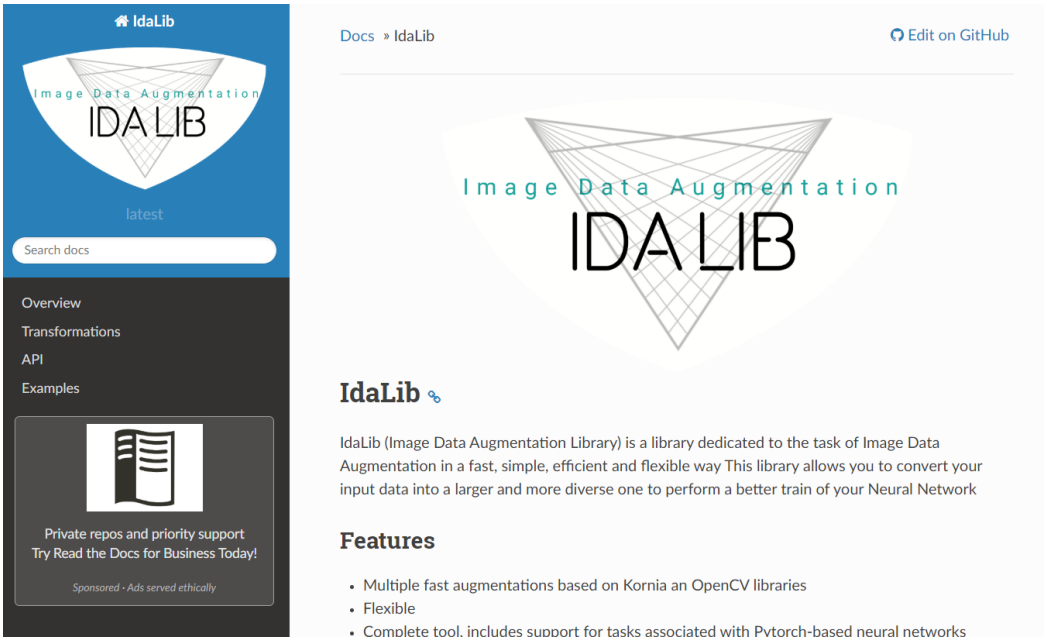



Figura 5.16: Interfaz de la documentación HTML de la librería.

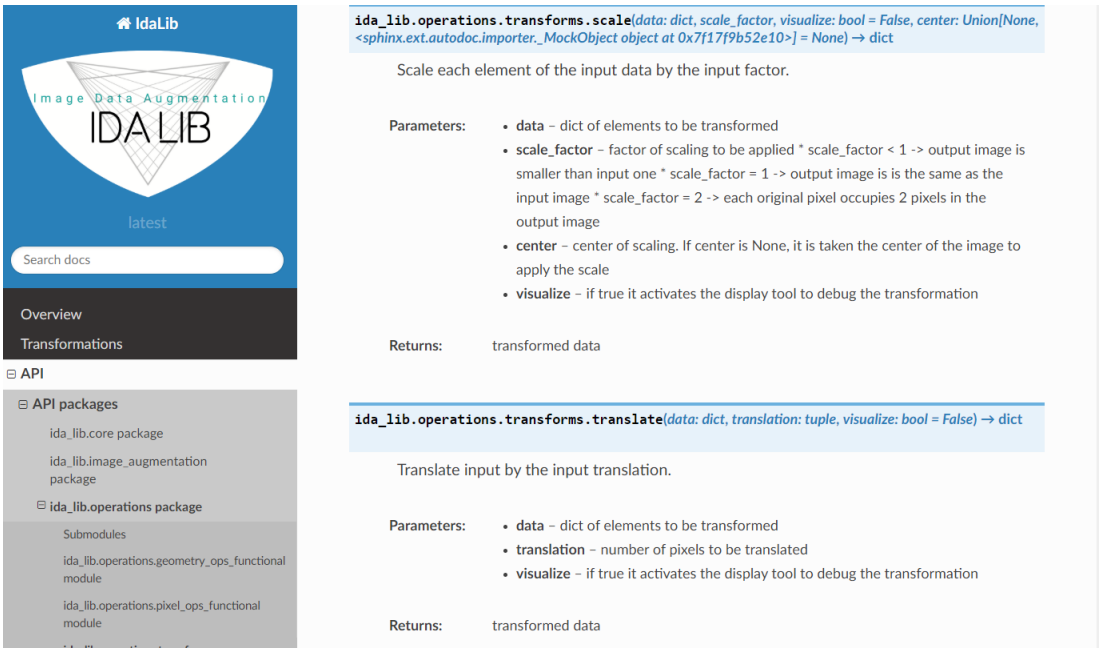



Figura 5.17: Interfaz de la documentación (ejemplo especificación API).

### 5.6.4. Publicación de la librería y documentación

Para la publicación del código de la librería se empleó la herramienta oficial de Python Packaging (PyPI) para que el paquete se pueda instalar mediante la utilidad `pip`. Se decidió este medio de instalación por ser el más sencillo para los usuarios, el más utilizado y que no requiere pasos intermedios para su instalación.

Para la publicación fue necesario adaptar el código a las estructuras establecidas por la organización PyPA en la guía *Python Packaging User Guide* [6], incluyendo archivos de configuración como: setup.py, requirements.txt y LICENSE. A continuación se generó la distribución fuente para finalmente subirlo al repositorio PyPI.

Una vez publicado el código de la API, para la publicación de la documentación (la generada en HTML con Sphinx) se emplea el repositorio gratuito ReadTheDocs. La documentación publicada se puede consultar en el enlace: `https://ida-lib.readthedocs.io/en/latest/index.html#`.

# Capítulo 6

# Verificación y validación

En este capítulo se describen todas las tareas del proyecto orientadas a asegurar que el software desarrollado está acorde a su especificación y cumple las necesidades de los clientes.

Como la metodología escogida para el proyecto es Scrum, cobra gran importancia el papel de la validación y verificación dado que en cada uno de los *sprints* se debe construir un elemento totalmente funcional y fácilmente ampliable.

## 6.1. Verificación

La verificación engloba las actividades de evaluación del sistema o componentes. Permite determinar si los productos de una determinada fase del desarrollo satisfacen las condiciones impuestas en el inicio de la etapa.

### 6.1.1. Plan de pruebas

Al emplear Scrum como metodología de desarrollo del proyecto, las pruebas se fueron realizando en cada uno de los *sprints* en los que se desarrollaba alguna funcionalidad. Al añadir nuevas funcionalidades en cada *sprint* se pueden introducir errores en las partes ya implementadas, por lo que se han desarrollado pruebas unitarias para comprobar el funcionamiento de elementos pequeños de código. Con este tipo de pruebas es más sencillo detectar dónde están los errores y, por lo tanto, se pueden corregir más rápido.

Para facilitar su comprensión, los casos de prueba se organizaron en los siguientes módulos:

- **Módulo de pruebas sobre transformaciones individuales:** las pruebas se realizan sobre las operaciones de transformaciones individuales.
- **Módulo de pruebas sobre el *pipeline*:** las pruebas se enfocan a verificar el correcto funcionamiento del *pipeline* ante diferentes parámetros y tipos de datos de entrada.

- **Módulo de pruebas sobre el sistema de entrada:** pruebas del funcionamiento del generador de datos (que integra el *pipeline*) ante diferentes parámetros.
- **Módulo de pruebas sobre el sistema de aumento a disco:** prueba del módulo de aumento de datos sobre disco.

La implementación de las pruebas se realizó con la herramienta **Pytest**, un framework de pruebas de Python orientado a su uso en APIs. Se definieron 23 pruebas diferentes siguiendo el esquema del Cuadro 6.1. Dado que cada prueba se realiza varias veces sobre diferentes operaciones o entradas como se indica en los cuadros siguientes, el número total de casos de prueba ascendió a 139.

Cuadro 6.1: Modelo de descripción y resultado de pruebas.

| **Código de la prueba-** | Nombre de la prueba |
|---|---|
| **Descripción** | Breve descripción del caso de prueba que representa. |
| **Entrada** | Detalle de los datos de entrada para la prueba. |
| **Resultado esperado** | Descripción del resultado que se espera de la ejecución de la prueba (en un caso de funcionamiento correcto). |
| **Resultado obtenido** | Correcto/incorrecto. |

### Módulo de pruebas sobre transformaciones individuales

Se realizan 8 operaciones de prueba, de la CP-001 a la CP-008, que se enumeran en las tablas siguientes.

| **CP-001**- Operaciones sobre imagen *numpy* | |
|---|---|
| **Descripción** | Se comprobará que cada una de las transformaciones unitarias se realiza sin errores sobre un elemento de entrada formado por una imagen de tipo *numpy*. |
| **Entrada** | ▪ *Numpy image:* diccionario Python con un único elemento que corresponde a una imagen de tipo *numpy* de 3 canales de tipo *uint8* (profundidad de color de 8 bits).<br>▪ Cada una de las operaciones individuales se aplica con parámetros válidos. |
| **Resultado esperado** | Diccionario Python con un único elemento que corresponde a una imagen de tipo *numpy* de 3 canales de tipo *uint8*. |
| **Resultado obtenido** | Correcto. |

| **CP-002**- Operaciones sobre imagen *numpy* en escala de grises | |
|---|---|
| **Descripción** | Se comprobará que cada una de las transformaciones unitarias se realiza sin errores sobre un elemento de entrada formado por una imagen en escala de grises (un único canal de color) de tipo *numpy*. |
| **Entrada** | ▪ *Numpy image:* diccionario Python con un único elemento que corresponde a una imagen de tipo *numpy* de 1 canal y de tipo *uint8* (profundidad de color de 8 bits).<br>▪ Cada una de las operaciones individuales se aplica con parámetros válidos. |
| **Resultado esperado** | Diccionario Python con un único elemento que corresponde a una imagen de tipo *numpy* de 1 canal de tipo *uint8*. |
| **Resultado obtenido** | Correcto. |

| **CP-003**- Operaciones sobre imagen tipo *torch-tensor* | |
|---|---|
| **Descripción** | Se comprobará que cada una de las transformaciones unitarias se realiza sin errores sobre un elemento de entrada formado por una imagen tipo *torch tensor* de 3 canales de color. |
| **Entrada** | ▪ *Torch image:* diccionario Python con un único elemento que corresponde a una imagen de tipo *torch tensor* de 3 canales y de tipo *uint8* (profundidad de color de 8 bits).<br>▪ Cada una de las operaciones individuales se aplica con parámetros válidos. |
| **Resultado esperado** | Diccionario Python con un único elemento que corresponde a una imagen de tipo *torch-tensor* de 3 canales y de tipo *uint8*. |
| **Resultado obtenido** | Correcto. |

| **CP-004**- Operaciones sobre elemento *numpy* compuesto | |
|---|---|
| **Descripción** | Se comprobará que cada una de las transformaciones unitarias se realiza sin errores sobre un elemento de entrada formado por una combinación de todos los posibles elementos de entrada (imagen, máscara, mapa de segmentación, coordenadas de puntos y mapa de calor). |
| **Entrada** | ▪ Elemento compuesto: diccionario Python con 6 elementos: imagen (de 3 canales) , mapa de calor, mapa de segmentación, máscara y lista de puntos, todos ellos de tipo *numpy* y *float32*, para comprobar el funcionamiento con diferentes profundidades de color.<br>▪ Cada una de las operaciones individuales se aplica con parámetros válidos. |
| **Resultado esperado** | Diccionario Python con los correspondientes 6 elementos de tipo *numpy* y *float32*. |
| **Resultado obtenido** | Correcto. |

| **CP-005**- Operaciones sobre elemento *torch-tensor* compuesto | |
|---|---|
| **Descripción** | Se comprobará que cada una de las transformaciones unitarias se realiza sin errores sobre un elemento de entrada formado por una combinación de todos los posibles elementos de entrada (imagen, máscara, mapa de segmentación, coordenadas de puntos y mapa de calor). |
| **Entrada** | ▪ Elemento compuesto: diccionario Python con 6 elementos: imagen (de 3 canales de color), mapa de calor, mapa de segmentación, máscara y lista de puntos, todos de tipo *torch-tensor* y *float32* para comprobar el funcionamiento con diferentes profundidades de color.<br>▪ Cada una de las operaciones individuales se aplica con parámetros válidos. |
| **Resultado esperado** | Diccionario Python con los correspondientes 6 elementos de tipo *torch-tensor* y *float32*. |
| **Resultado obtenido** | Correcto. |

| **CP-006**- Operaciones a nivel de píxel sobre elemento de entrada sin imagen | |
|---|---|
| **Descripción** | Se comprobará que se gestiona de manera adecuada el error sobre cada una de las operaciones a nivel de píxel. |
| **Entrada** | ▪ Elemento compuesto sin imagen: diccionario Python con 2 elementos: mapa de segmentación y máscara, ambos de tipo *numpy* y *float32*.<br>▪ Cada una de las operaciones a nivel de píxel individuales se aplica con parámetros válidos. |
| **Resultado esperado** | Lanzamiento de una excepción correspondiente a la falta de la imagen en el elemento de entrada. |
| **Resultado obtenido** | Correcto. |

| **CP-007**- Operaciones geométricas sobre elemento de entrada sin imagen | |
|---|---|
| **Descripción** | Comprobar que se transforma de manera correcta y sin errores los elementos de entrada que carecen de imagen entre sus elementos. |
| **Entrada** | ▪ Elemento compuesto sin imagen: diccionario Python con 2 elementos: mapa de segmentación y máscara, ambos de tipo *numpy* y *float32*.<br>▪ Cada una de las operaciones geométricas individuales se aplica con parámetros válidos. |
| **Resultado esperado** | Ejecución sin errores. Salida de un diccionario Python con los correspondientes 2 elementos de tipo *numpy* y *float32*. |
| **Resultado obtenido** | Correcto. |

| **CP-008**- Operaciones geométricas sobre elemento con componentes múltiples de un mismo tipo | |
|---|---|
| **Descripción** | Comprobar que se transforman de manera correcta y sin errores los elementos de entrada que contienen más de un elemento de cada tipo (por ejemplo, que incluyan 2 máscaras diferentes). |
| **Entrada** | ▪ Elemento compuesto con 2 máscaras: diccionario Python con 2 máscaras ('mask1' y 'mask2'), ambas de tipo *numpy* y *float32*.<br>▪ Cada una de las operaciones geométricas individuales se aplica con parámetros válidos. |
| **Resultado esperado** | Ejecución sin errores. Salida de un diccionario Python con los correspondientes 2 elementos de tipo *numpy* y *float32*. |
| **Resultado obtenido** | Correcto. |

### Módulo de pruebas sobre el *pipeline*

Se realizan 11 operaciones de prueba, de la CP-009 a la CP-019, que se enumeran en las tablas siguientes.

| **CP-009**- Ejecución del *pipeline* sobre un lote de imágenes *numpy* | |
|---|---|
| **Descripción** | Comprobar la correcta ejecución del pipeline sobre un lote de imágenes. |
| **Entrada** | ▪ Lote de imágenes: lista de diccionarios Python compuestos por una única imagen de tipo *numpy* y *uint8*.<br>▪ *Pipeline* genérico con los parámetros por defecto y conjunto operaciones genéricas. |
| **Resultado esperado** | Ejecución sin errores y de salida una lista de diccionarios Python de un único elemento de tipo *torch-tensor* y *uint8*. |
| **Resultado obtenido** | Correcto. |

| **CP-010**- Ejecución del *pipeline* con operación de *resize* | |
|---|---|
| **Descripción** | Comprobar que el *pipeline* se ejecuta sin errores al incluir la operación de redimensionado de los elementos de entrada. Comprobar que los elementos de salida tienen el tamaño indicado. |
| **Entrada** | ▪ Lote de imágenes: lista de diccionarios Python compuestos por una única imagen de tipo *numpy* y *uint8*.<br>▪ *Pipeline* genérico con los parámetros por defecto y conjunto operaciones genéricas.<br>▪ Valor de redimensionado (10,20). |
| **Resultado esperado** | Ejecución sin errores. Salida de una lista de diccionarios Python de un único elemento de tipo *torch-tensor* y *uint8* con el tamaño indicado en el *resize* (10,20). |
| **Resultado obtenido** | Correcto. |

| **CP-011**- Ejecución del *pipeline* sobre un lote de elementos sin imagen | |
|---|---|
| **Descripción** | Comprobar que el *pipeline* se ejecuta sin errores al emplear un lote de elementos que no incluyen imágenes. |
| **Entrada** | ▪ Lote de elementos sin imagen: lista de diccionarios Python compuestos por una máscara y un mapa de segmentación *numpy* y *uint8*.<br>▪ *Pipeline* genérico con los parámetros por defecto y conjunto operaciones genéricas. |
| **Resultado esperado** | Ejecución sin errores. Salida de una lista de diccionarios Python con los correspondientes 2 elementos de tipo *torch-tensor* y *uint8*. |
| **Resultado obtenido** | Correcto. |

| **CP-012**- Ejecución del *pipeline* con lista de operaciones vacía | |
|---|---|
| **Descripción** | Comprobar que el *pipeline* se ejecuta sin errores al no incluir operaciones de transformación. |
| **Entrada** | ▪ Lote de imágenes: lista de diccionarios Python compuestos por una única imagen de tipo *numpy* y *uint8*.<br>▪ *Pipeline* genérico con parámetros por defectos y lista de operaciones vacía. |
| **Resultado esperado** | Ejecución sin errores. Salida de una lista de diccionarios Python con imágenes de tipo *torch-tensor*. |
| **Resultado obtenido** | Correcto. |

| **CP-013**- Ejecución del *pipeline* con todas las operaciones | |
|---|---|
| **Descripción** | Comprobar que el *pipeline* se ejecuta sin errores al incluir todas las operaciones posibles. Comprobar también que todas las operaciones se ejecutan sin errores. |
| **Entrada** | ▪ Lote de imágenes: lista de diccionarios Python compuestos por una única imagen de tipo *numpy* y *uint8*.<br>▪ *Pipeline* genérico con parámetros por defecto y lista de operaciones con todas las operaciones del *pipeline*. |
| **Resultado esperado** | Ejecución sin errores. Salida de una lista de diccionarios Python con imágenes de tipo *torch-tensor*. |
| **Resultado obtenido** | Correcto. |

| **CP-014**- Ejecución del *pipeline* con mapeo de puntos en operación de *flip* | |
|---|---|
| **Descripción** | Comprobar la ejecución del *pipeline* incluyendo operaciones de reflexión (*flip* vertical o *flip* horizontal) con mapeo de puntos. Comprobar también el correcto mapeo de puntos. |
| **Entrada** | ▪ Lote de imágenes y coordenadas de puntos: lista de diccionarios Python compuestos por una imagen de tipo *numpy* y *uint8* y una lista de $n$ coordenadas de puntos.<br>▪ *Pipeline* con los parámetros por defecto y 2 operaciones de *flip* horizontal, una de ellas con mapeo de puntos. Tras 2 *flip* horizontales la imagen debe ser idéntica a la original, permitiendo comparar fácilmente el mapeo de puntos. Mapeo de prueba: ((0,1) , (3,4)). |
| **Resultado esperado** | Ejecución sin errores. Salida de un diccionario Python con los correspondientes 2 elementos de tipo *numpy* y *float32*. Las coordenadas son intercambiadas correctamente después de la transformación, respetando el orden indicado como parámetro. |
| **Resultado obtenido** | Correcto. |

| **CP-015**- Ejecución del *pipeline* con lote de elementos compuestos | |
|---|---|
| **Descripción** | Comprobar la ejecución correcta del pipeline con lote de entrada que incluye todos los tipos de objetos (imagen, máscara, mapa de segmentación, mapa de calor y coordenadas de puntos) para verificar el correcto tratamiento de cada tipo de datos. |
| **Entrada** | ▪ Elemento compuesto: lista de diccionarios Python con imagen, máscara, mapa de calor, mapa de segmentación y coordenadas de puntos.<br>▪ *Pipeline* genérico con los parámetros por defecto y conjunto operaciones genéricas. |
| **Resultado esperado** | Ejecución sin errores. Salida de una lista de diccionarios Python con los correspondientes elementos. |
| **Resultado obtenido** | Correcto. |

| **CP-016**- Ejecución del *pipeline* con lote de elementos con múltiples componentes de un mismo tipo | |
|---|---|
| **Descripción** | Comprobar la ejecución correcta del pipeline con lote de entrada que incluye varios elementos de un mismo tipo de dato. |
| **Entrada** | ▪ Lote de elementos con 2 máscaras: lista de diccionarios Python con 2 máscaras ('mask1' y 'mask2') ambos de tipo *numpy* y *float32*.<br>▪ *Pipeline* genérico con los parámetros por defecto y conjunto operaciones genéricas. |
| **Resultado esperado** | Ejecución sin errores. Salida de un diccionario Python con las correspondientes 2 máscaras ('mask1' y 'mask2'). |
| **Resultado obtenido** | Correcto. |

| **CP-017**- Ejecución del *pipeline* con parámetros de entrada válidos | |
|---|---|
| **Descripción** | Comprobar la correcta ejecución del *pipeline* con diferentes combinaciones de parámetros válidos de entrada. |
| **Entrada** | Parámetros válidos del *pipeline*:<br>▪ Interpolation: 'nearest', 'bilinear'.<br>▪ Padding_mode: 'zeros', 'reflection', 'border'.<br>▪ Output_format: 'dict', 'tuple'. |
| **Resultado esperado** | Ejecución sin errores. |
| **Resultado obtenido** | Correcto. |

| **CP-018**- Ejecución del *pipeline* con parámetros de entrada erróneos | |
|---|---|
| **Descripción** | Comprobar que se maneja de manera correcta la entrada de parámetros no válidos. |
| **Entrada** | ▪ *Pipeline* genérico con parámetros por defecto y conjunto de operaciones genéricas.<br>▪ Valores incorrectos para: 'interpolation', 'padding_mode', 'output_format'. |
| **Resultado esperado** | Lanzamiento de una excepción asociada al valor incorrecto de los parámetros. |
| **Resultado obtenido** | Correcto. |

| **CP-019**- Selección del tipo de dato de salida en el *pipeline* | |
|---|---|
| **Descripción** | Comprobar que la salida del *pipeline* cumple con el tipo de dato indicado como parámetro. |
| **Entrada** | ▪ Lote de imágenes y coordenadas de puntos: lista de diccionarios Python compuestos por una imagen de tipo *numpy* y *uint8* y una lista de $n$ coordenadas de puntos.<br>▪ *Pipeline* genérico con valores para el parámetro 'output_type': torch.uint8, torch.int16, torch.int32, torch.int64, torch.float16, torch.float32, torch.float64. |
| **Resultado esperado** | Ejecución sin errores. Salida de un diccionario Python con imágenes cuyo tipo es igual al indicado en 'output_type'. |
| **Resultado obtenido** | Correcto. |

### Módulo de pruebas sobre el sistema de entrada

Se realizan 2 operaciones de prueba, de la CP-020 a la CP-021, que se enumeran en las tablas siguientes.

| **CP-020**- Ejecución del *dataloader* normal | |
|---|---|
| **Descripción** | Comprobar que se ejecuta de manera correcta el objeto *dataloader* de la librería, con cualquier valor de *batchsize*. |
| **Entrada** | ▪ *Dataset* de entrada simple que genera elementos *numpy* de valores aleatorios formando diccionarios de imágenes y máscaras.<br>▪ Tamaño de lote: valores 1, 2, 3, 5 y 10. |
| **Resultado esperado** | Ejecución sin errores. Salida de un *dataloader* iterable que genera lotes de $n$ elementos (del formato del *dataset* imagen-máscara) concatenados, donde $n$ tiene que coincidir con el tamaño de lote indicado. |
| **Resultado obtenido** | Correcto. |

| **CP-021**- Ejecución del *dataloader* con operación de *resize* | |
|---|---|
| **Descripción** | Comprobar que el *dataloader* se construye y ejecuta sin errores incluyendo una operación de *resize* en sus elementos. |
| **Entrada** | ▪ *Dataset* de entrada simple que genera elementos *numpy* de valores aleatorios formando diccionarios de imágenes y máscaras.<br>▪ *resize* parámetro de redimensionado que tomará diferentes valores: (10, 10), (10, 50), (50, 10), (500, 500). |
| **Resultado esperado** | Ejecución sin errores. *Dataloader* iterable que genera lotes de $n$ elementos (del formato del *dataset* imagen-máscara) concatenados, donde $n$, que es el tamaño de las imágenes y máscaras de salida, coincida con el tamaño indicado como *resize*. |
| **Resultado obtenido** | Correcto. |

**Módulo de pruebas sobre el sistema de aumento a disco**

Se realizan 2 operaciones de prueba, de la CP-022 a la CP-023, que se enumeran en las tablas siguientes.

| **CP-022**- Ejecución de *image data augmentation* a disco | |
|---|---|
| **Descripción** | Ejecución del objeto *AugmentToDisk* para comprobar la correcta ejecución del *image data augmentation* sobre disco. |
| **Entrada** | ▪ *Dataset* de entrada simple que genera elementos *numpy* de valores aleatorios formando diccionarios de imágenes y máscaras.<br>▪ Lista genérica de operaciones de transformación.<br>▪ Parámetros del objeto *AugmentToDisk* por defecto.<br>▪ *Samples per item*: número de elementos generados a disco por cada elemento de entrada. Se probarán los valores: 1, 2, 5 y 10. |
| **Resultado esperado** | Ejecución sin errores, que incluye la creación de un directorio con la ruta indicada en *output_path* y el número de elementos generados en ese directorio debe ser igual a<br><br>Samples per item $\times$ 2 $\times$ número elementos *dataset*<br><br>(La multiplicación por 2 se debe a que cada elemento se corresponde con una imagen y una máscara). |
| **Resultado obtenido** | Correcto. |

| **CP-023**- Ejecución de *image data augmentation* a disco con selección de la extensión | |
|---|---|
| **Descripción** | Ejecución del objeto *AugmentToDisk* para comprobar la correcta ejecución del *image data augmentation* sobre disco, seleccionando la extensión para los elementos a guardar en el directorio. |
| **Entrada** | ▪ *dataset* de entrada simple que genera elementos *numpy* de valores aleatorios formando diccionarios de imágenes y máscaras.<br>▪ Lista genérica de operaciones de transformación.<br>▪ Parámetros del objeto *AugmentToDisk* por defecto.<br>▪ *Output_extension*: extensión deseada para los elementos de salida. Se probarán los valores: '.jpg', '.png' y '.jpeg' |
| **Resultado esperado** | Ejecución sin errores y creación de elementos en el directorio indicado y con la extensión indicada. |
| **Resultado obtenido** | Correcto. |

## 6.2. Validación

La fase de validación del producto trata de verificar que el producto construido cumple con lo esperado por el cliente. Debido a la naturaleza del sistema y al estado de alarma del COVID-19 la validación se llevará a cabo mediante la revisión de cumplimiento de los criterios de aceptación fijados para el proyecto y un estudio de rendimiento del sistema.

### 6.2.1. Criterios de aceptación

En este apartado se realiza una revisión de los criterios de aceptación del producto a nivel global (mencionados en la sección 3.2.2) y un análisis breve sobre su estado actual (de superación o no superación).

| **CA.01** - Correcta ejecución del *image data augmentation* sobre cualquier tipo de dato | |
|---|---|
| **Descripción** | La herramienta permite la realización del proceso de *image data augmentation* sobre conjuntos de imágenes de cualquier tipo y tamaño sin generar errores (propios del código). |
| **Estado** | **Superado**, se ha verificado su funcionamiento sobre los distintos tipos de datos de entrada con las pruebas unitarias. |

| **CA.02** - Variabilidad de operaciones de transformación | |
|---|---|
| **Descripción** | La herramienta ofrece un repertorio de transformaciones igual o superior al ofrecido por herramientas similares (10 o más operaciones). |
| **Estado** | **Superado**, el número de operaciones ofrecidas por la librería es de 17 sin tener en cuenta las operaciones aleatorizadas, con las que la cifra ascendería a 24 operaciones. |

| **CA.03** - Soporte de metadatos y cualquier combinación de los mismos | |
|---|---|
| **Descripción** | La herramienta soporta transformaciones conjuntas sobre imágenes, mapas de calor o saliencia, máscaras, mapas de segmentación, coordenadas de puntos y cualquier combinación de los mismos de manera correcta y consistente. |
| **Estado** | **Superado**, se ha verificado su funcionamiento sobre distintas combinaciones de metadatos con las pruebas unitarias. |

| **CA.04** - Herramienta de visualización | |
|---|---|
| **Descripción** | La herramienta de visualización se despliega con éxito y presenta los datos de manera correcta. |
| **Estado** | **Superado**, la herramienta de visualización se despliega con éxito y sin generar errores. |

| **CA.05** - *Image data augmentation* sobre disco y memoria volátil | |
|---|---|
| **Descripción** | La herramienta permite la realización del *image data augmentation* tanto sobre disco como sobre memoria volátil. |
| **Estado** | **Superado**, el módulo de *image data augmentation* sobre disco se ejecuta con éxito y ha sido verificado en las pruebas. Y tanto el *pipeline* como el *dataloader* de IDALib permiten la realización de *image data augmentation* sobre memoria volátil (ambos testeados también en el plan de pruebas). |

| **CA.06** - Estandarización de la librería | |
|---|---|
| **Descripción** | La herramienta puede considerarse una librería escalable, que respeta los estándares del lenguaje y las buenas prácticas asociadas, siguiendo la guía de estilo de Python **PEP8** para asegurar un código limpio y legible en la medida de lo posible. |
| **Estado** | **Superado**, el último *sprint* de desarrollo se dedicó en exclusiva a esta tarea, estandarizando el código (PEP 8), la estructura de la librería (Python Packaging User Guide), los *doscstrings* en formato *reStructuredText* y su documentación. |

### 6.2.2. *Benchmarking* del sistema

La motivación del proyecto es la construcción de una herramienta de *image data augmentation* que cubra las necesidades detectadas en el sector orientándolo a obtener el mejor rendimiento y eficiencia posible.

Para medir en qué medida se consiguió, se realizaron una serie de pruebas de rendimiento o *benchmarks* con el objeto de analizar el comportamiento del sistema ante diferentes entradas. La realización de todas las pruebas se llevó a cabo en el equipo de desarrollo, que muestra las siguientes características:

- GPU NVIDIA® GeForce ® GTX1050
- Intel Core i7-8750H (6 núcleos)

Las condiciones generales sobre las que se realizan todas las pruebas son:

- Las pruebas se realizan sobre el *pipeline*, por ser el elemento central de la API y del que depende el rendimiento del resto de funciones de la misma.
- Las diferentes transformaciones pueden tener diferentes costes temporales (no es lo mismo cambiar el brillo a una imagen que aplicarle una transformación afín). Por ello se crea una lista con todas las operaciones y, cada vez que se ejecuta el experimento, se toma una muestra aleatoria de $n$ operaciones de esta lista.
- Cada experimento se repite 10 veces (tomando cada vez un conjunto de operaciones diferentes) y se promedian los 10 resultados para evitar distorsiones por la selección específica de operaciones.

En el *image data augmentation* por lo general nunca se hacen transformaciones individuales, sino que se aplica una composición de transformaciones, por lo que es necesario analizar cómo escala el rendimiento del *pipeline* según el número de operaciones a implementar.

En el Cuadro 6.2 podemos observar los tiempos que requiere el *pipeline* para realizar las transformaciones variando el número de operaciones de 1 a 14. Para contrastar la optimización realizada se añade la columna *Tiempo secuencial*, que muestra el tiempo que requeriría el número de operaciones indicado si se aplicasen las transformaciones de manera secuencial:

$$TiempoSecuencial = NumeroOperaciones \times TiempoOperacion$$

Cuadro 6.2: Tiempos (en milisegundos) de transformación de IDALib variando el número de operaciones del *pipeline* sobre imágenes de dimensiones 750x750x1. La columna Tiempo medio de la composición representa el tiempo medio de transformación por elemento de entrada. En la de Tiempo secuencial se muestra el tiempo que requeriría el número de operaciones indicado si se aplicasen las transformaciones de manera secuencial.

| **Nº de operaciones** | **Tiempo medio de la composición** | **Tiempo secuencial** |
|---|---|---|
| 1 | 4.603 | 4.603 |
| 2 | 4.576 | 9.207 |
| 3 | 4.600 | 13.810 |
| 4 | 4.602 | 18.413 |
| 5 | 4.600 | 23.016 |
| 6 | 4.608 | 27.620 |
| 7 | 4.618 | 32.223 |
| 8 | 4.652 | 36.826 |
| 9 | 4.651 | 41.429 |
| 10 | 4.674 | 46.033 |
| 11 | 4.687 | 50.636 |
| 12 | 4.715 | 55.239 |
| 13 | 4.717 | 59.843 |
| 14 | 4.749 | 64.446 |

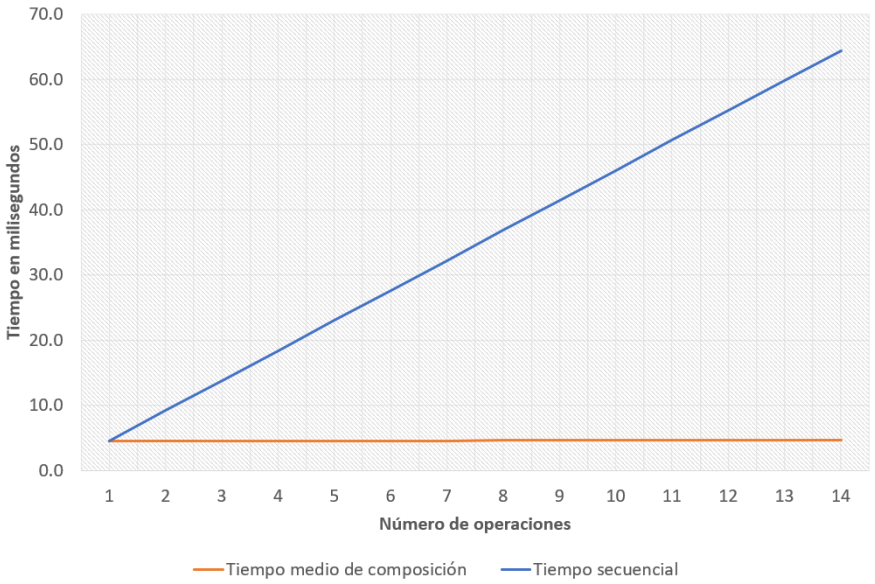



Figura 6.1: Comparación de tiempo medio de composición con IDALib y tiempo secuencial según el número de operaciones.

Como se puede ver en la Figura 6.1, la forma en la que escala el sistema con el número de operaciones es realmente bueno. El comportamiento que presenta puede ser considerado uniforme independientemente del número de operaciones incluidas, ya que por ejemplo existe una diferencia de menos de 0,15 milisegundos de media entre aplicar 1 operación y aplicar 15.

Otro de los aspectos a evaluar es el modo en que escala la librería según el número de metadatos que incluyen los elementos de entrada. Es decir, cómo cambia su rendimiento entre trabajar, por ejemplo, con 1 mapa de calor y trabajar con 15 mapas de calor.

En el Cuadro 6.3 podemos ver la variación de tiempo entre transformar datos compuestos por una única máscara a transformar datos de hasta 10 máscaras. Como en la prueba anterior, se dispone una columna (Tiempo secuencial) que representa el tiempo necesario para las $n$ máscaras si se procesasen una a una de manera secuencial.

Cuadro 6.3: Tiempos (en milisegundos) de transformación (compuesta por 5 operaciones[1]) de IDALib variando el número de máscaras (de dimensiones 750x750x1) que componen el elemento de entrada. En la columna Tiempo medio de la composición se indica el tiempo medio de transformación por elemento de entrada (compuesto por el número de máscaras indicado). En la de Tiempo secuencial se muestra el tiempo que requeriría el número de de operaciones indicado si se aplicasen las transformaciones de manera secuencial a cada máscara.

| **Nº de máscaras** | **Tiempo medio de la composición** | **Tiempo secuencial** |
|---|---|---|
| **1** | 4.831 | 4.831 |
| **2** | 7.452 | 9.662 |
| **3** | 6.763 | 14.493 |
| **4** | 8.694 | 19.324 |
| **5** | 11.625 | 24.155 |
| **6** | 14.986 | 28.986 |
| **7** | 19.027 | 33.817 |
| **8** | 23.228 | 38.648 |
| **9** | 27.489 | 43.479 |
| **10** | 32.140 | 48.314 |

[1]La transformación que se aplica a cada elemento está compuesta por 5 operaciones individuales de transformación.

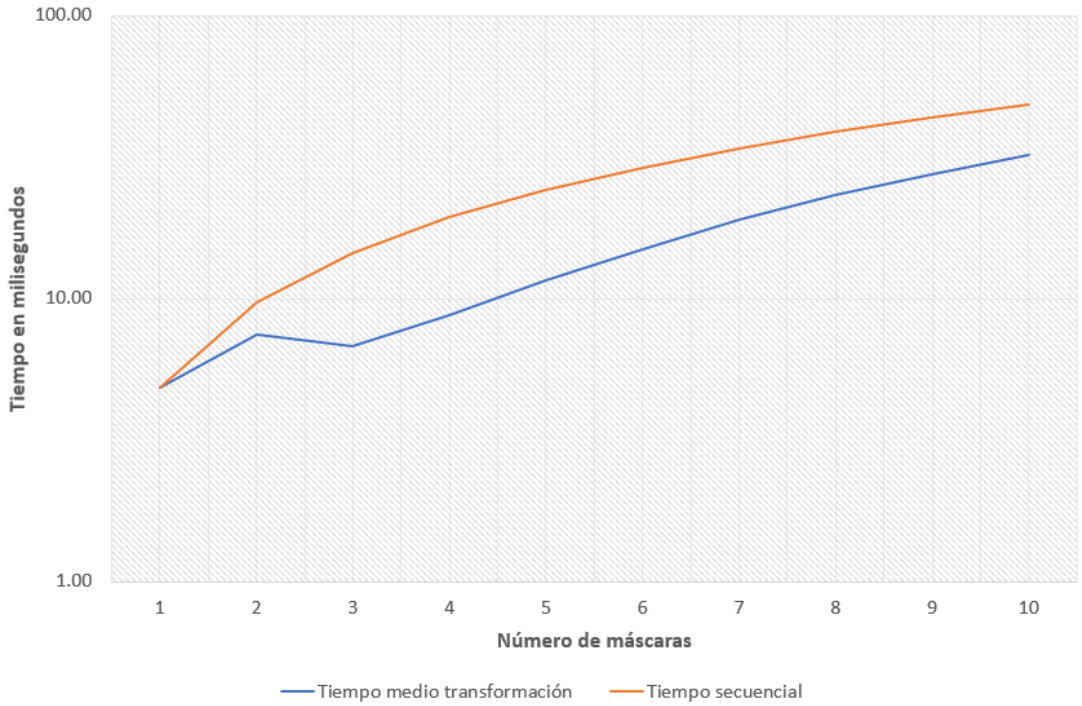



Figura 6.2: Comparación del tiempo de transformación de IDALib con el tiempo secuencial variando el número de máscaras que componen el elemento de entrada.

Como se ve en la Figura 6.2, el escalado con número de metadatos presenta cierta optimización. A pesar de que la capacidad de escalar no es tan buena como en el caso anterior, sigue siendo una medida positiva si tenemos en cuenta que la transformación de una máscara es prácticamente igual que la transformación de una imagen y, estas operaciones se realizan mediante librerías a bajo nivel tan optimizadas que hacen complicado conseguir mejorar más el rendimiento. Las medidas sobre máscaras pueden ser representativas de otros metadatos como son los mapas de calor o los mapas de segmentación.

Los metadatos del tipo puntos reciben un tratamiento diferente a los de tipo máscara. Por esta razón es necesario evaluar el comportamiento del sistema según el número de puntos a transformar. En el Cuadro 6.4 podemos ver la diferencia de tiempo entre que el *pipeline* transforme un elemento con un punto, hasta el tiempo para transformar elementos con 1000 puntos. En este caso no se incluye referencia al tiempo secuencial ya que se mide la transformación de coordenadas de puntos acompañadas de una imagen (no se tiene referencia directa del tiempo que requiere la transformación de un punto).

Cuadro 6.4: Tiempos (en milisegundos) de transformación de IDALib variando el número de puntos del elemento de entrada. En la columna Tiempo medio se representa el tiempo medio de transformación por elemento de entrada (compuesto por el número de puntos correspondiente).

| **Nº de puntos** | **Tiempo medio** |
|---|---|
| 1 | 4.752 |
| 60 | 4.663 |
| 120 | 4.677 |
| 180 | 4.654 |
| 240 | 4.718 |
| 300 | 4.660 |
| 360 | 4.671 |
| 420 | 4.843 |
| 480 | 4.669 |
| 540 | 4.737 |
| 600 | 4.876 |
| 660 | 4.690 |
| 720 | 4.703 |
| 780 | 4.678 |
| 840 | 4.679 |
| 900 | 4.673 |
| 960 | 4.678 |
| 1000 | 4.669 |

En el caso de los puntos, el escalado vuelve a ser óptimo. Vemos en la Cuadro 6.4 que, usando IDALib, prácticamente lleva el mismo tiempo la transformación de 1 punto que la de 1000 puntos.

Con estas 3 pruebas, se puede afirmar que la API construida es escalable.

Por último, para tener una referencia de la mejora que produce la composición de operaciones en IDALib, se compara su rendimiento temporal en la transformación con composición de operaciones respecto a las librerías de bajo nivel.

En el Cuadro 6.5 podemos ver los tiempos de transformación de IDALib frente a las librerías a bajo nivel valoradas en apartados anteriores. Para realizar estas medidas se implementa una transformación compuesta por 5 operaciones diferentes. En el caso de IDALib, se realiza mediante el *pipeline* (que realiza la composición de operaciones). Para Kornia y OpenCV estas 5 operaciones se realizan de manera secuencial (no ofrecen composición de operaciones). Para NVIDIA DALI se implementa la composición de operaciones al emplear también su propio *pipeline*.

Cuadro 6.5: Tiempos (en milisegundos) de transformación por imagen (con 5 operaciones) de las librerías variando la resolución de la imagen de entrada.

| | **IDALib** | **Kornia GPU** | **OpenCV CPU** | **OpenCV GPU** | **NVIDIA DALI** |
|---|---|---|---|---|---|
| **50x50** | 1.598 | 5.293 | 15.228 | 16.750 | 1.776 |
| **100x100** | 1.618 | 5.391 | 15.940 | 18.331 | 1.855 |
| **250x250** | 1.758 | 6.803 | 27.063 | 21.650 | 1.869 |
| **500x500** | 2.716 | 12.460 | 48.960 | 36.720 | 1.981 |
| **750x750** | 4.679 | 22.186 | 57.521 | 50.909 | 2.800 |
| **1000x1000** | 7.594 | 35.731 | 64.415 | 58.658 | 4.700 |

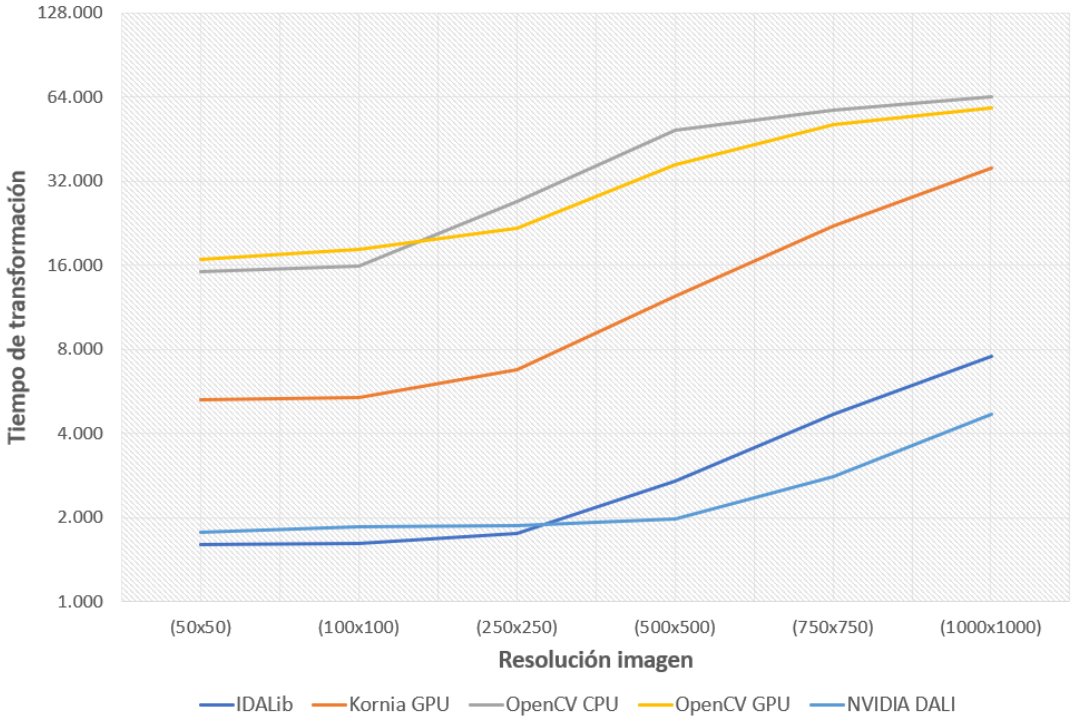



Figura 6.3: Comparación del tiempo de transformación (con 5 operaciones) de las librerías variando la resolución de la imagen de entrada.

La composición de operaciones hace que IDALib se coloque como una de las mejores opciones. Pese a que NVIDIA DALI presenta mejores resultados para tamaños más grandes de imagen, IDALIb sigue ofreciendo el soporte a metadatos y otras funcionalidades que estas librerías no ofrecen. La diferencia de tiempo con respecto a las otras librerías es garantía de que IDALib es una herramienta eficiente y por tanto que cumple con los objetivos del cliente.

# Capítulo 7

# Conclusión y posibles ampliaciones

El objetivo de este proyecto fue el desarrollo de IDALib, una librería para *image data augmentation*. Este proyecto surge como una necesidad en la realización de trabajos de aprendizaje profundo en los que se participó. En ellos se detectó la importancia de esta tarea para el entrenamiento de la red dado que el *image data augmentation* consumía gran parte de la potencia de computación. Por todo esto, era importante analizar y optimizar este proceso para hacerlo más ligero.

A pesar de las condiciones extraordinarias causadas por el COVID-19, este proyecto se llevó sin mayores inconvenientes y fue una oportunidad para experimentar las complicaciones y ventajas que supone el teletrabajo. Como el proyecto integraba una parte de investigación y una parte de desarrollo, supuso un reto mayor pero también una oportunidad de afrontar un proyecto completo e interdisciplinar.

El proyecto pudo adaptarse de forma sencilla a las condiciones extraordinarias causadas por el estado de alarma. A lo largo del mismo, se llevó a cabo una tarea de investigación inicial sobre las distintas opciones para el desarrollo de la librería. En la práctica, esta parte de investigación no solo permitió seleccionar las librerías para el desarrollo, sino que obligó a aprender a usar diferentes herramientas, pudiendo analizar sus diferencias, ventajas y carencias. Este conocimiento de las herramientas del sector facilitó notablemente la toma de decisiones posterior, ya que ayudaba a comprender las necesidades del mercado y las alternativas que ofrecía.

En lo que se refiere al desarrollo, dados los resultados que se obtuvieron del *benchmarking* en la validación, se puede concluir que todas las decisiones tomadas y optimizaciones implementadas fueron adecuadas. La librería final cubre unas necesidades del sector que otras alternativas no ofrecen y presenta un rendimiento y escalabilidad muy competitivos.

Finalmente, la publicación de la librería y de su documentación hicieron que este proyecto pueda ser utilizado por otros usuarios para agilizar sus experimentos. De manera global es satisfactorio crear, desde cero, un proyecto completo y acabar poniéndolo a la disposición de la comunidad.

Aportar a la comunidad el trabajo desarrollado es una manera de compensar y agradecer todos los trabajos de otros usuarios que se emplearon y que fueron tantas veces de gran ayuda. De no ser por ellos, no se podría haber construido esta herramienta.

## 7.1. Posibles ampliaciones

Una librería por lo general es el resultado de un trabajo de mantenimiento y mejora continuo que se va reflejando en las nuevas versiones o *releases*. Como tal, IDALib tiene muchas posibilidades de crecimiento y ampliaciones.

Entre las ampliaciones sería importante añadir elementos que se dejaron fuera del alcance por la limitación temporal de este proyecto, como por ejemplo la incorporación de un mayor repertorio de operaciones de transformación que también son comunes en proyectos de aprendizaje profundo (*random erasing*, corrección HUE, fenómenos atmosféricos, transformaciones elásticas...).

Pese que se abordaron los metadatos más comunes, sería positivo también incorporar soporte para cajas delimitadoras (*bounding boxes*) muy comunes en redes neuronales para la identificación de elementos en imágenes.

Además, la estructura de procesado del *pipeline* debe ser revisada y analizada para identificar posibles optimizaciones, como por ejemplo el uso de otras librerías a bajo nivel.

En esta primera versión se priorizó el rendimiento temporal de la API. Pero también sería importante analizar el uso de memoria de la librería, detectando sus carencias (si hay) e implementando mejoras.

En general, el objetivo es construir una librería flexible y que se adapte a las necesidades de los usuarios. Por ello, la librería aceptará como ampliación cualquier necesidad detectada en base al *feedback* de los usuarios, así como posibles funcionalidades del sector que no hayan quedado cubiertas.

# Apéndice A

# Manual IDALib

A continuación se incluye una sección del manual de IDALib. Este manual se ha generado de manera automática con la herramienta Sphinx e incluye la documentación completa de la librería. Dado que es el idioma más extendido para la realización de publicaciones científicas, el manual se encuentra escrito en inglés. En la presente memoria solo se incluye una parte del mismo y en el formato algunas líneas aparecen cortadas, por lo que pido disculpas, pero el manual está completamente disponible en la web. Se puede acceder a la documentación completa de IDALib en el siguiente enlace:
`https://ida-lib.readthedocs.io/en/latest/index.html`

El código de la API se puede consultar en:

- `https://github.com/raquelvilas18/ida_lib`
- `https://nubeusc-my.sharepoint.com/:f:/g/personal/raquel_rodriguez_vilas_rai_usc_es/EhwRMwqFWxxDpt6I_u3tODwBAmfpMeUN1vfWO8gppmYtqQ?e=moeE17`

CHAPTER

TWO

# INSTALATION

You can use pip to install Ida-Lib

```
pip install ida-lib
```

If you want to get the latest version of the code before it is released on PyPI you can install the library from GitHub:

```
pip install -U git+https://github.com/raquelvilas18/ida_lib
```

## 3.3 API

### 3.3.1 API packages

#### ida_lib.core package

#### Submodules

#### ida_lib.core.pipeline module

**class** ida_lib.core.pipeline.**Pipeline**(*pipeline_operations: list*, *resize: tuple = None*, *interpolation: str = 'bilinear'*, *padding_mode: str = 'zeros'*, *output_format: str = 'dict'*, *output_type: Optional[torch.dtype] = None*)

Bases: object

The pipeline object represents the pipeline with data transformation operations (pictures, points). When executed, on a batch of images,it applies the necessary transformations (being different on each image based on the probabilities of each operation included).

**Considerations:** 1) The images must be of the same size, or the RESIZE operation must be included so that the transformations can be applied correctly 2) To run the pipeline, it accepts any type of input metadata named in the input dict. In particular it gives special treatment

**to data named as:**

- Mask: it is affected by geometric transformations and its output is discrete to values of 0-1
- Segmap: generalization of mask. Every value is discrete
- Image: affected by geometric and color transformations
- Keypoints: geometric transformations are applied to them as coordinates.
- Others: any other metadata will not be transformed (example: 'tag', 'target'...)

Example:

**pip = pipeline(resize = (25, 25), pipeline_operations=(** translate_pipeline(probability=0.5, translation=(3, 0.05)), vflip_pipeline(probability=0.5), hflip_pipeline(probability=0.5), contrast_pipeline(probability=0.5, contrast_factor=1), random_brightness_pipeline(probability=0.2, brightness_range=(1.5, 1.6)), random_scale_pipeline(probability=1, scale_range=(0.5, 1.5), center_deviation=20), random_rotate_pipeline(probability=0.2, degrees_range=(-50, 50), center_deviation=20))

))

**get_data_types**() → tuple

Returns the tuple of data types identified on the input data

### ida_lib.core.pipeline_functional module

ida_lib.core.pipeline_functional.**get_compose_matrix**(*operations: list*, *data_info: Optional[dict] = None*) → None._VariableFunctionsClass.tensor

Returns the transformation matrix composed by the multiplication in order of the input operations (according to their probability) If data_info is not None, go through the operations by entering the necessary information about the images (image center, shape..)

**Parameters**

- **operations** – list of pipeline operations
- **data_info** – dict with data info to configure operations parameters

**Returns** torch tensor of the transform matrix

ida_lib.core.pipeline_functional.**get_compose_function**(*operations: list*) → numpy.ndarray

returns the LUT table with the correspondence of each possible value according to the color operations to be implemented (according to their probability)

**Parameters** **operations** – list of pipeline operations

**Returns** compose function

ida_lib.core.pipeline_functional.**preprocess_data**(*data: Union[list, dict]*, *batch_info: Union[list, dict] = None*, *interpolation: str = None*, *resize: Optional[tuple] = None*) → list

Combines the 2d information in a tensor and the points in a homogeneous coordinate matrix that allows applying the geometric operations in a single joint operation on the data and another on the points.

- Loads the data as tensor in GPU to prepare them as input to a neural network
- Analyze the data info required for the transformations (shape, bpp. . . )
- Resize the 2d data and keypoints to the new shape

**Parameters**

- **resize** – if it is wanted to resize the data, indicate the new size
- **data** – list of elements to be transformed through the pipe
- **batch_info** – dict with the required data info
- **interpolation** – desired interpolation mode to be applied

**Returns** preprocessed and resized data, and dict with batch info

ida_lib.core.pipeline_functional.**split_operations_by_type**(*operations: list*) → tuple

Split input operations into sub-lists of each transformation type the normalization operation is placed last to apply correctly the other operations

**Parameters** **operations** – list of pipeline operations

**Returns** tuple of lists of the operations separated into color, geometry and independent

ida_lib.core.pipeline_functional.**postprocess_data**(*batch: list*, *batch_info: dict*, *data_original: Optional[list] = None*, *visualize: bool = False*, *original_type: torch.dtype = torch.uint8*, *output_format: str = 'dict'*) → list

Restores the data to the original form; separating the matrix into the different 2d input data and point coordinates.

**Parameters**

- **batch** – list of elements to be transformed through the pipe
- **batch_info** – dict with necessary information about the batch data
- **data_original** – original batch before transforms
- **visualize** – whether to run the visualization tool or not
- **original_type** – torch original type of the input data to do the conversion of the output data to this type
- **output_format** – whether to format the output element as a dict or as a tuple

**Returns** processed data

ida_lib.core.pipeline_functional.**switch_point_positions**(*point_matrix*, *input_list*)

### ida_lib.core.pipeline_geometric_ops module

**class** ida_lib.core.pipeline_geometric_ops.**HflipPipeline**(*probability: float = 1*, *exchange_points: tuple = None*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Horizontally flip the input image-mask-keypoints and 2d data

**config_parameters**(*data_info: dict*)

**get_op_matrix**() → None._VariableFunctionsClass.tensor

**need_data_info**() → bool

**switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**VflipPipeline**(*probability: float*, *exchange_points: tuple = None*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Vertically flip the input image-mask-keypoints and 2d data

**config_parameters**(*data_info: dict*)

**get_op_matrix**() → None._VariableFunctionsClass.tensor

**need_data_info**() → bool

**switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**RotatePipeline**(*degrees: int*, *center: None._VariableFunctionsClass.tensor = None*, *probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Rotate the input image-mask-keypoints and 2d data by the input degrees

**config_parameters**(*data_info: dict*)

**get_op_matrix**() → None._VariableFunctionsClass.tensor

**need_data_info**() → bool

**static switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**ShearPipeline**(*shear: tuple*, *probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Shear the input image-mask-keypoints and 2d data by the input shear factor

**get_op_matrix**() → None._VariableFunctionsClass.tensor

**static need_data_info**() → bool

**static switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**ScalePipeline**(*scale_factor: Union[float, tuple]*, *probability: float = 1*, *center: None._VariableFunctionsClass.tensor = None*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Scale the input image-mask-keypoints and 2d data by the input scaling value

**config_parameters**(*data_info: dict*)

**get_op_matrix**()

**need_data_info**()

**static switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**TranslatePipeline**(*translation: tuple*, *probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Translate the input image-mask-keypoints and 2d data by the input translation

**get_op_matrix**() → None._VariableFunctionsClass.tensor

**static need_data_info**() → bool

**static switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**RandomScalePipeline**(*probability: float*, *scale_range: tuple*, *keep_aspect: bool = True*, *center_deviation: int = None*, *center: None._VariableFunctionsClass.tensor = None*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Scale the input image-mask-keypoints and 2d data by a random scaling value calculated within the input range

**config_parameters**(*data_info: dict*)

**get_op_matrix**() → None._VariableFunctionsClass.tensor

**need_data_info**() → bool

**static switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**RandomRotatePipeline**(*degrees_range: tuple*, *probability: float = 1*, *center_deviation: int = None*, *center: None._VariableFunctionsClass.tensor = None*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Rotate the input image-mask-keypoints and 2d data by a random scaling value calculated within the input range

**config_parameters**(*data_info: dict*)

**get_op_matrix**() → None._VariableFunctionsClass.tensor

**need_data_info**() → bool

**static switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**RandomShearPipeline**(*probability: float*, *shear_range: tuple*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Shear the input image-mask-keypoints and 2d data by a random shear value calculated within the input range

**get_op_matrix**() → None._VariableFunctionsClass.tensor

**static need_data_info**() → bool

**static switch_points**()

**class** ida_lib.core.pipeline_geometric_ops.**RandomTranslatePipeline**(*probability: float*, *translation_range: tuple*, *same_translation_on_axis: bool = False*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Translate the input image-mask-keypoints and 2d data by a random translation value calculated within the input range

**get_op_matrix**()

**static need_data_info**() → bool

**static switch_points**()

### ida_lib.core.pipeline_local_ops module

**class** ida_lib.core.pipeline_local_ops.**BlurPipeline**(*probability: float = 1*, *blur_size: tuple = 5, 5*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Blur input image ( non-weighted blur)

**apply_to_image_if_probability**(*img: numpy.ndarray*) → numpy.ndarray

**get_op_matrix**()

**class** ida_lib.core.pipeline_local_ops.**GaussianBlurPipeline**(*probability: float = 1*, *blur_size: tuple = 5, 5*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Blur input image by a Gaussian function

**apply_to_image_if_probability**(*img: numpy.ndarray*) → numpy.ndarray

**get_op_matrix**()

**class** ida_lib.core.pipeline_local_ops.**GaussianNoisePipeline**(*probability: float = 1*, *var: float = 0.5*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Add gaussian noise to the input image (gaussian noise is a statistical noise having a probability density function (PDF) equal to that of the normal distribution)

**apply_to_image_if_probability**(*img: numpy.ndarray*) → numpy.ndarray

**get_op_matrix**()

**class** ida_lib.core.pipeline_local_ops.**PoissonNoisePipeline**(*probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Add poison noise to the input image (Speckle is a granular interference that inherently exists in and degrades the quality of the active radar, synthetic aperture radar (SAR), medical ultrasound and optical coherence tomography images. It is applied by adding Poisson-distributed noise)

**apply_to_image_if_probability**(*img: numpy.ndarray*) → numpy.ndarray

**get_op_matrix**()

**class** ida_lib.core.pipeline_local_ops.**SaltAndPepperNoisePipeline**(*probability=1*, *amount: Optional[float] = 0.01*, *s_vs_p: Optional[float] = 0.5*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Add salt and pepper noise to the input image (salt-and-pepper noise is a statistical noise compose of white (salt) and black (pepper) pixels)

**apply_to_image_if_probability**(*img: numpy.ndarray*) → numpy.ndarray

**get_op_matrix**()

**class** ida_lib.core.pipeline_local_ops.**SpekleNoisePipeline**(*probability: float = 1*, *mean: Optional[float] = 0*, *var: Optional[float] = 0.01*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Add spekle noise to the input image (Speckle is a granular interference that inherently exists in and degrades the quality of the active radar, synthetic aperture radar (SAR), medical ultrasound and optical coherence tomography images. It is applied by adding the image multiplied by the noise matrix -> img + img * uniform_noise)

**apply_to_image_if_probability**(*img: numpy.ndarray*) → numpy.ndarray

**get_op_matrix**()

### ida_lib.core.pipeline_operations module

**class** ida_lib.core.pipeline_operations.**PipelineOperation**(*op_type: str*, *probability: float = 1*)

Bases: abc.ABC

Abstract class of pipeline operations

**apply_according_to_probability**() → bool

**abstract get_op_matrix**()

**get_op_type**()

### ida_lib.core.pipeline_pixel_ops module

**class** ida_lib.core.pipeline_pixel_ops.**ContrastPipeline**(*contrast_factor: float*, *probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change the contrast of the input image.

**get_op_matrix**()

**transform_function**(*x: int*) → float

**class** ida_lib.core.pipeline_pixel_ops.**RandomContrastPipeline**(*contrast_range: tuple*, *probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change the contrast of the input image with a random contrast factor calculated within the input range

**get_op_matrix**()

**transform_function**(*x*)

**class** ida_lib.core.pipeline_pixel_ops.**BrightnessPipeline**(*brightness_factor: float*, *probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change brightness of the input image

**get_op_matrix**()

**get_op_type**()

**transform_function**(*x: int*) → float

**class** ida_lib.core.pipeline_pixel_ops.**RandomBrightnessPipeline**(*probability: float*, *brightness_range: tuple*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change brightness of the input image to random amount calculated within the input range

**get_op_matrix**()

**get_op_type**()

**transform_function**(*x: int*) → float

**class** ida_lib.core.pipeline_pixel_ops.**GammaPipeline**(*gamma_factor: float*, *probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change the luminance of the input image

**get_op_matrix**()

**transform_function**(*x: int*) → float

**class** ida_lib.core.pipeline_pixel_ops.**RandomGammaPipeline**(*gamma_range: tuple*, *probability: float = 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change the luminance of the input image by a random gamma factor calculated within the input range

**get_op_matrix**()

**transform_function**(*x: int*) → float

**class** ida_lib.core.pipeline_pixel_ops.**NormalizePipeline**(*probability: float = 1*, *old_range: tuple = 0, 255*, *new_range: tuple = 0, 1*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change the pixels value to a normalize range

**get_op_matrix**()

**static transform_function**(*x: int*) → float

**class** ida_lib.core.pipeline_pixel_ops.**DenormalizePipeline**(*probability: float = 1*, *old_range: tuple = 0, 1*, *new_range: tuple = 0, 255*)

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Denormalize pixel value

**get_op_matrix**()

**transform_function**(*x: int*) → float

### Module contents

### ida_lib.image_augmentation package

### Submodules

### ida_lib.image_augmentation.augment_to_disk module

**class** ida_lib.image_augmentation.augment_to_disk.**AugmentToDisk**(*dataset: torch.utils.data.dataset.Dataset*, *samples_per_item: Optional[int] = 2*, *total_output_samples: Optional[int] = None*, *operations: Optional[list] = None*, *interpolation: str = 'bilinear'*, *padding_mode: str = 'zeros'*, *resize: Optional[tuple] = None*, *output_extension: str = '.jpg'*, *output_csv_path: str = 'annotations.csv'*, *output_path: str = './augmented'*)

Bases: object

The AugmentToDisk object allows to perform Data Image Augmentation directly to disk. That is, to save the images generated to disk to be used in future processes.

**final_save**()

**This method can be overwritten to make a customized saving of the items according to the interests of the user**

Method that runs only once, once all the images have been processed. Useful for writing csv with image annotations. By default the annotations of all images are saved in the same file. The csv file will have one row for each generated element, identified by its id. Each column will correspond with the labels associated to each generated element. In the case of coordinate lists, their coordinates are arranged in columns separating the x and y coordinates in each element (point0_x, point0_y, point1_x, . . . , point_y)

**save_item**(*item: dict*, *index: int*, *output_path: str*, *types_2d: list*, *other_types: list*, *element: int*)

**This method can be overwritten to make a customized saving of the items according to the interests of the user**

Method that implements the way to save to disk each of the generated elements. By default it saves all the generated images in the specified path. The samples are organized by name following the form:

- images: <id_image>_<sample number> <extension>
- other two-dimensional types: <id_image>-<data_type>_<sample number> <extension>

Annotations on the data, such as labels, or point coordinates are stored in dictionaries that will be written when all the images have been processed.

**Parameters**

- **item** – input element to be saved to disk
- **element** – input element number to identify it
- **index** – sample number to which the input item corresponds
- **output_path** – path to the directory in which to save the generated data
- **types_2d** – list of types of two dimensional data of the input item
- **other_types** – list of types that are not two-dimensional elements

### ida_lib.image_augmentation.data_loader module

**class** ida_lib.image_augmentation.data_loader.**AugmentDataLoader**(*batch_size*, *dataset: torch.utils.data.dataset.Dataset*, *shuffle=True*, *pipeline_operations=None*, *resize=None*, *interpolation: str = 'bilinear'*, *padding_mode: str = 'zeros'*, *output_format: str = 'dict'*, *output_type: Optional[torch.dtype] = None*)

Bases: torch.utils.data.dataloader.DataLoader

**The DataAugmentDataLoader class implements a Pytorch DataLoader but groups it into one class:**

- The Dataset object that takes care of reading the data
- The iterative DataLoader object that will serve as an input system for a neural network.
- A pipeline that applies data image Augmentation operations over the input data.

To make use of this class, it is necessary to provide a dataset to make a personalized reading of your data.

**class InnerDataset**(*pipeline*, *dataset*)

Bases: torch.utils.data.dataset.Dataset

inner dataset is an internal class that uses the DataAugmentDataLoader to add the pipeline to the input dataset

### Module contents

### ida_lib.operations package

### Submodules

### ida_lib.operations.geometry_ops_functional module

ida_lib.operations.geometry_ops_functional.**affine_compose_data**(*data: dict*, *matrix: None._VariableFunctionsClass.tensor*) → dict

**Parameters**

- **data** – dict of elements to be transformed
- **matrix** – matrix of transformation

**Returns** transformed data

ida_lib.operations.geometry_ops_functional.**affine_coordinates_matrix**(*matrix_coordinates: None._VariableFunctionsClass.t ma-trix_transformation: None._VariableFunctionsClass.t* → None._VariableFunctionsClass.t

ida_lib.operations.geometry_ops_functional.**affine_image**(*img: None._VariableFunctionsClass.tensor*, *matrix: None._VariableFunctionsClass.tensor*, *interpolation: str = 'bilin-ear'*, *padding_mode: str = 'zeros'*) → None._VariableFunctionsClass.tensor

ida_lib.operations.geometry_ops_functional.**config_scale_matrix**(*scale_factor*, *center*, *matrix*)

ida_lib.operations.geometry_ops_functional.**get_rotation_matrix**(*center: None._VariableFunctionsClass.tensor*, *degrees: None._VariableFunctionsClass.tensor*)

ida_lib.operations.geometry_ops_functional.**get_scale_matrix**(*center: None._VariableFunctionsClass.tensor*, *scale_factor: Union[float, None._VariableFunctionsClass.tensor]*)

ida_lib.operations.geometry_ops_functional.**get_shear_matrix**(*shear_factor: tuple*) → None._VariableFunctionsClass.tensor

ida_lib.operations.geometry_ops_functional.**get_squared_scale_matrix**(*center: None._VariableFunctionsClass.te scale_factor: Union[float, None._VariableFunctionsClass.te*

ida_lib.operations.geometry_ops_functional.**get_squared_shear_matrix**(*shear_factor: tuple*) → None._VariableFunctionsClass.te

ida_lib.operations.geometry_ops_functional.**get_translation_matrix**(*translation: None._VariableFunctionsClass.tenso*) → None._VariableFunctionsClass.tenso

ida_lib.operations.geometry_ops_functional.**hflip_compose_data**(*data: dict*) → dict

**Parameters** **data** – dict of elements to be transformed

**Returns** transformed data

ida_lib.operations.geometry_ops_functional.**hflip_coordinates_matrix**(*matrix: None._VariableFunctionsClass.te width: int*) → None._VariableFunctionsClass.te

ida_lib.operations.geometry_ops_functional.**hflip_image**(*img: None._VariableFunctionsClass.tensor*) → None._VariableFunctionsClass.tensor

ida_lib.operations.geometry_ops_functional.**own_affine**(*tensor: torch.Tensor, matrix: torch.Tensor, interpolation: str = 'bilinear', padding_mode: str = 'border'*) → torch.Tensor

Apply an affine transformation to the image.

**Parameters**

- **tensor** – The image tensor to be warped.
- **matrix** – The 2x3 affine transformation matrix.
- **interpolation** – interpolation mode to calculate output values 'bilinear' | 'nearest'. Default: 'bilinear'.
- **padding_mode** – padding mode for outside grid values 'zeros' | 'border' | 'reflection'. Default: 'zeros'.

**Returns** The warped image.

ida_lib.operations.geometry_ops_functional.**prepare_data**(*func*)

Decorator that prepares the input data to apply the geometric transformation. For this purpose, it concatenates all the two-dimensional elements of the input data in the same tensor on which a single transformation is applied. If the input data contains point coordinates, they are grouped in a matrix as homogeneous coordinates, over which a single matrix multiplication is performed.

**Parameters** **func** – geometric function to be applied to the data

**Returns** processed data

ida_lib.operations.geometry_ops_functional.**rotate_compose_data**(*data: dict*, *degrees: None._VariableFunctionsClass.tensor*, *center: None._VariableFunctionsClass.tensor*)

**Parameters**

- **data** – dict of elements to be transformed
- **degrees** – counterclockwise degrees of rotation
- **center** – center of rotation. Default, center of the image

**Returns** transformed data

ida_lib.operations.geometry_ops_functional.**rotate_coordinates_matrix**(*matrix_coordinates: None._VariableFunctionsClass.*
*matrix: None._VariableFunctionsClass.*
→ None._VariableFunctionsClass.t

ida_lib.operations.geometry_ops_functional.**rotate_image**(*img: None._VariableFunctionsClass.tensor*, *degrees: None._VariableFunctionsClass.tensor*, *center: None._VariableFunctionsClass.tensor*)
→ None._VariableFunctionsClass.tensor

mode

ida_lib.operations.geometry_ops_functional.**scale_compose_data**(*data: dict*, *scale_factor: Union[float, None._VariableFunctionsClass.tensor]*, *center: Optional[None._VariableFunctionsClass.tens* = *None*) → dict

**Parameters**

- **data** – dict of elements to be transformed
- **scale_factor** – factor of scaling
- **center** – center of scaling. By default its taken the center of the image

**Returns** transformed data

ida_lib.operations.geometry_ops_functional.**scale_coordinates_matrix**(*matrix_coordinates: None._VariableFunctionsClass.te*
*matrix: None._VariableFunctionsClass.te*
→ None._VariableFunctionsClass.te

ida_lib.operations.geometry_ops_functional.**scale_image**(*img: None._VariableFunctionsClass.tensor*, *scale_factor: None._VariableFunctionsClass.tensor*, *center: None._VariableFunctionsClass.tensor*) → None._VariableFunctionsClass.tensor

ida_lib.operations.geometry_ops_functional.**shear_compose_data**(*data: dict*, *shear_factor: tuple*) → dict

**Parameters**

- **data** – dict of elements to be transformed
- **shear_factor** – pixels of shearing

**Returns** transformed data

ida_lib.operations.geometry_ops_functional.**shear_coordinates_matrix**(*matrix_coordinates: None._VariableFunctionsClass.te*
*matrix: None._VariableFunctionsClass.te*
→ None._VariableFunctionsClass.te

ida_lib.operations.geometry_ops_functional.**shear_image**(*img: None._VariableFunctionsClass.tensor*, *shear_factor: None._VariableFunctionsClass.tensor*) → None._VariableFunctionsClass.tensor

ida_lib.operations.geometry_ops_functional.**translate_compose_data**(*data: dict*, *translation: Union[int, None._VariableFunctionsClass.tenso*
→ dict

**Parameters**

- **data** – dict of elements to be transformed
- **translation** – number of pixels to translate

**Returns** transformed data

ida_lib.operations.geometry_ops_functional.**translate_coordinates_matrix**(*matrix_coordinates: None._VariableFunctionsC*
*translation: None._VariableFunctionsC*
→ None._VariableFunctionsC

ida_lib.operations.geometry_ops_functional.**translate_image**(*img: None._VariableFunctionsClass.tensor*, *translation: None._VariableFunctionsClass.tensor*) → None._VariableFunctionsClass.tensor

ida_lib.operations.geometry_ops_functional.**vflip_compose_data**(*data: dict*) → dict

**Parameters** **data** – dict of elements to be transformed

**Returns** transformed data

ida_lib.operations.geometry_ops_functional.**vflip_coordinates_matrix**(*matrix: None._VariableFunctionsClass.te*, *height: int*) → None._VariableFunctionsClass.te

ida_lib.operations.geometry_ops_functional.**vflip_image**(*img: None._VariableFunctionsClass.tensor*) → None._VariableFunctionsClass.tensor

### ida_lib.operations.pixel_ops_functional module

ida_lib.operations.pixel_ops_functional.**apply_blur**(*img*, *blur_size=5, 5*)

ida_lib.operations.pixel_ops_functional.**apply_gaussian_blur**(*img*, *blur_size=5, 5*)

**Parameters**

- **img** – input image to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**apply_gaussian_noise**(*image*, *var=20*)

ida_lib.operations.pixel_ops_functional.**apply_lut_by_pixel_function**(*function*, *image: numpy.ndarray*) → numpy.ndarray

Applies the input operation to the image using a LUT

**Parameters**

- **function** – Mathematical function that represents the operation to carry out in each pixel of the image
- **image** – input image

**Returns**

ida_lib.operations.pixel_ops_functional.**apply_poisson_noise**(*image*)

ida_lib.operations.pixel_ops_functional.**apply_salt_and_pepper_noise**(*image*, *amount=0.05*, *s_vs_p=0.5*)

ida_lib.operations.pixel_ops_functional.**apply_spekle_noise**(*image*, *mean=0*, *var=0.01*)

ida_lib.operations.pixel_ops_functional.**blur**(*img: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*, *blur_size*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

**Parameters**

- **img** – input image to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**change_brightness**(*image: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*, *brightness: int*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

Change the brightness of the input image.

**Parameters**

- **image** – input image to be normalized
- **brightness** – desired amount of brightness for the image 0 - no brightness 1 - same 2 - max brightness

**Returns** transformed image

ida_lib.operations.pixel_ops_functional.**change_contrast**(*image: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*, *contrast*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

:param image : input image to be transformed :param contrast: modification factor to be applied to the image contrast

- 0 - total contrast removal
- 1 - dont modify
- >1 - augment contrast

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**change_gamma**(*image: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*, *gamma: float*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

**Parameters**

- **image** – input image to be transformed

- **gamma** – desired factor gamma * gamma = 0 -> removes image luminance (black output image) * gamma = 1 -> remains unchanged * gamma > 1 -> increases luminance

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**gaussian_blur**(*img: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*, *blur_size*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

**Parameters**

- **img** – input image to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**gaussian_noise**(*image: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*, *var=20*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

**Parameters**

- **image** – input image to be transformed
- **var** – var of the gaussian distribution of noise

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**get_brightness_function**(*brightness: int*)

**Parameters** **brightness** – brightness factor

**Returns** Return the lambda function of the brightness operation

ida_lib.operations.pixel_ops_functional.**get_contrast_function**(*contrast: float*)

**Parameters** **contrast** – modification factor to be applied to the image contrast

**Returns** Return the lambda function of the contrast operation

ida_lib.operations.pixel_ops_functional.**get_gamma_function**(*gamma*)

**Parameters** **gamma** – desired factor gamma

**Returns** Returns the lambda function of the gamma adjust operation

ida_lib.operations.pixel_ops_functional.**histogram_equalization**(*img: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

**Parameters** **img** – input image to be transformed

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**normalize_image**(*img: numpy.ndarray*, *norm_type: int = 32*) → numpy.ndarray

Normalize the input image

**Parameters**

- **img** – input image to be normalized
- **norm_type** – opencv normalization type (' cv2.NORM_MINMAX' |cv2.NORM_HAMMING |cv2.NORM_HAMMING2 |cv2.NORM_INF |cv2.NORM_RELATIVE . . . )

**Returns** normalized image

ida_lib.operations.pixel_ops_functional.**poisson_noise**(*image: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

**Parameters** **image** – input image to be transformed

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**prepare_data_for_opencv**(*func*)

Decorator that prepares the input data to apply the transformation that only affects the image (color). :param func: color function to be applied to the data :return: processed data

ida_lib.operations.pixel_ops_functional.**salt_and_pepper_noise**(*image: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*, *amount*, *s_vs_p*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

**Parameters**

- **image** – input image to be transformed
- **amount** – percentage of image's pixels to be occupied by noise
- **s_vs_p** – percentage of salt respect total noise. Default same salt (white pixel) as pepper (black pixels)

**Returns** returns the transformed image

ida_lib.operations.pixel_ops_functional.**spekle_noise**(*image: Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]*, *mean=0*, *var=0.01*) → Union[dict, None._VariableFunctionsClass.tensor, numpy.ndarray]

**Parameters**

- **image** – input image to be transformed
- **mean** – mean of noise distribution
- **var** – variance of noise distribution

**Returns** returns the transformed image

### ida_lib.operations.transforms module

ida_lib.operations.transforms.**hflip**(*data: dict*, *visualize: bool = False*) → dict

Horizontally flip the input data.

**Parameters**

- **data** – dict of elements to be transformed
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**vflip**(*data: dict*, *visualize: bool = False*) → dict

Vertically flip the input data.

**Parameters**

- **data** – dict of elements to be transformed
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**affine**(*data: dict*, *matrix: None._VariableFunctionsClass.tensor*, *visualize: bool = False*) → dict

Applies affine transformation to the data

:param data dict of elements to be transformed :param matrix: matrix of transformation :param visualize: if true it activates the display tool to debug the transformation :return: transformed data

ida_lib.operations.transforms.**rotate**(*data: dict*, *degrees: float*, *visualize: bool = False*, *center: Union[None, torch.Tensor] = None*) → dict

Rotate each element of the input data by the indicated degrees counterclockwise

**Parameters**

- **data** – dict of elements to be transformed
- **degrees** – degrees of rotation
- **center** – center of rotation. If center is None, it is taken the center of the image to apply the rotation
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**shear**(*data: dict*, *shear_factor: tuple*, *visualize: bool = False*) → dict

Shear input data by the input shear factor

**Parameters**

- **data** – dict of elements to be transformed

- **shear_factor** – pixels to shear
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**scale**(*data: dict*, *scale_factor*, *visualize: bool = False*, *center: Union[None, torch.Tensor] = None*) → dict

Scale each element of the input data by the input factor.

**Parameters**

- **data** – dict of elements to be transformed
- **scale_factor** – factor of scaling to be applied * scale_factor < 1 -> output image is smaller than input one * scale_factor = 1 -> output image is is the same as the input image * scale_factor = 2 -> each original pixel occupies 2 pixels in the output image
- **center** – center of scaling. If center is None, it is taken the center of the image to apply the scale
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**translate**(*data: dict*, *translation: tuple*, *visualize: bool = False*) → dict

Translate input by the input translation.

**Parameters**

- **data** – dict of elements to be transformed
- **translation** – number of pixels to be translated
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**change_gamma**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *gamma*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Adjust image's gamma (luminance correction) . if the input data is a dictionary, only those corresponding

to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **gamma** – desired gamma factor (luminance of image)
  - gamma = 0 -> removes image luminance (black output image)
  - gamma = 1 -> remains unchanged
  - gamma > 1 -> increases luminance
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**change_contrast**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *contrast: float*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Change the image contrast. if the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **contrast** – desired contrast factor to the data * contrast = 0 -> removes image contrast (white output image) * contrast = 1 -> remains unchanged * contrast > 1 -> increases contrast

:param visualize : if true it activates the display tool to debug the transformation :return: transformed data

ida_lib.operations.transforms.**change_brightness**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *bright=1*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Change the image brightness. if the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **bright** – desired brightness amount for the data
  - brightness = 0 -> removes image brightness (black output image)
  - brightness = 1 -> remains unchanged
  - brightness > 1 -> increases brightness
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**equalize_histogram**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Equalize image histogram. if the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**inject_gaussian_noise**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *var=0.5*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Inject gaussian noise. If the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **var** – variance of the noise distribution
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**inject_poisson_noise**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Inject poisson noise. if the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**inject_spekle_noise**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *mean=0*, *var=0.01*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Inject poisson noise. if the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **mean** – mean of noise distribution
- **var** – variance of noise distribution
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**inject_salt_and_pepper_noise**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *amount=0.05*, *s_vs_p=0.5*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Inject salt and pepper noise if the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **amount** – percentage of image's pixels to be occupied by noise

:param s_vs_p : noise type distribution. Default same salt (white pixel) as pepper (black pixels) :param visualize: if true it activates the display tool to debug the transformation :return: transformed data

ida_lib.operations.transforms.**blur**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *blur_size=5, 5*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Blur image. if the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

ida_lib.operations.transforms.**gaussian_blur**(*data: Union[dict, torch.Tensor, numpy.ndarray]*, *blur_size=5, 5*, *visualize: bool = False*) → Union[dict, torch.Tensor, numpy.ndarray]

Blurring an image by a Gaussian function. if the input data is a dictionary, only those corresponding to an image are altered

**Parameters**

- **data** – dict of elements to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)
- **visualize** – if true it activates the display tool to debug the transformation

**Returns** transformed data

### ida_lib.operations.utils module

ida_lib.operations.utils.**add_new_axis**(*arr: numpy.ndarray*)

ida_lib.operations.utils.**arrays_equal**(*arr1*, *arr2*)

ida_lib.operations.utils.**data_to_numpy**(*data*)

ida_lib.operations.utils.**dtype_to_torch_type**(*im_type: numpy.dtype*)

Maps the numpy type to the equivalent torch.type

**Parameters** **im_type** – numpy type

**Returns** torch.type

ida_lib.operations.utils.**element_to_dict_csv_format**(*item*, *name*)

ida_lib.operations.utils.**get_principal_type**(*data: dict*)

ida_lib.operations.utils.**get_torch_image_center**(*data*)

ida_lib.operations.utils.**homogeneous_points_to_list**(*keypoints*)

ida_lib.operations.utils.**homogeneous_points_to_matrix**(*keypoints*)

ida_lib.operations.utils.**is_a_normalized_image**(*image*)

ida_lib.operations.utils.**is_numpy_data**(*data*)

ida_lib.operations.utils.**keypoints_to_homogeneous_and_concatenate**(*keypoints*, *resize_factor=None*)

ida_lib.operations.utils.**keypoints_to_homogeneous_functional**(*keypoints*)

ida_lib.operations.utils.**map_value**(*x*, *in_min*, *in_max*, *out_min*, *out_max*)

ida_lib.operations.utils.**mask_change_to_01_functional**(*mask*)

ida_lib.operations.utils.**remove_digits**(*label: str*)

ida_lib.operations.utils.**round_torch**(*arr: None._VariableFunctionsClass.tensor*, *n_digits: int = 3*)

ida_lib.operations.utils.**save_im**(*tensor*, *title*)

#### Module contents

### 3.3.2 Submodules

### 3.3.3 ida_lib.visualization module

ida_lib.visualization.**visualize**(*images: dict*, *images_originals: dict*, *max_images: int = 5*)
Generate the bokeh plot of the input batch transformation

**Parameters**

- **images** – list of transformed items (dict of image and other objects)
- **images_originals** – list of original items (dict of image and other objects)
- **max_images** – max number of tabs to be shown

ida_lib.visualization.**plot_image_transformation**(*data*, *data_original*)
Generate the bokeh plot of the input batch transformation

**Parameters**

- **data** – input dict element
- **data_original** – original input element (before transforms)

## 3.4 Examples

You can find all the examples here: https://github.com/raquelvilas18/ida_lib/tree/master/examples

### 3.4.1 Pipeline usage example

#### Pipeline Usage example

```
"""In this file an example of how to use the idaLib pipeline is shown, in which you␣
↪can see:
* how to declare the pipeline
* which format to use for the input elements
* how to display or not the results
* and how to execute it in general.
```

(continues on next page)

# Apéndice B

# Licencia de la API

# Bibliografía


[1] Project Management Institute (PMI). *Guía PMBOK*. Quinta edición. 2013.

[2] Sommerville, Ian. *Ingeniería del Software*. Pearson Education, 2005.

[3] Sebastian Raschka. *Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow Packt Publishing*; Second Edition.

[4] Connor Shorten and Taghi M. Khoshgoftaar, *A survey on Image Data Augmentation For Deep Learning*.

[5] A. Fawzi, H. Samulowitz, D. Turaga and P. Frossard, *Adaptive data augmentation for image classification*, 2016 IEEE International Conference on Image Processing (ICIP), Phoenix.

[6] PyPA: Python Packaging User Guide, 2020.
Disponible en *https://packaging.python.org/*.

[7] August Ferdinand Möbius, *El cálculo baricéntrico (Der barycentrische Calcul)*.

[8] Arthur Coste.Image Processing : *Affine Transformation, Landmarks registration, Non linear Warping*.

[9] Uilin Liu and Kevin J. Shih and Ting-Chun Wang and Fitsum A. Reda and Karan Sapra and Zhiding Yu and Andrew Tao and Bryan Catanzaro *Image padding modes (Partial Convolution based Padding)*. (2018)

[10] Vishnu Subramanian. *Deep Learning with PyTorch: A Practical Approach to Building Neural Network Models Using PyTorch*. (2018).

[11] Ekin D. Cubuk and Barret Zoph and Dandelion Mane and Vijay Vasudevan and Quoc V. Le. *AutoAugment: Learning Augmentation Strategies using Backpropagation*. (2018)

[12] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, Hideki Nakayama. *Faster AutoAugment: Learning Augmentation Strategies using Backpropagation*. (2019)

[13] Connor Shorten and Taghi M. Khoshgoftaar, *A survey on Image Data Augmentation For Deep Learning*.J Big Data 6, 60 (2019).

[14] Ekin D. Cubuk and Barret Zoph and Jonathon Shlens and Quoc V. Le. *Randaugment: Practical automated data augmentationwith a reduced search space*. (2019).

[15] Nicolás Vila Blanco, Maria José Carreira Nouche, Paulina Varas Quintana. *Deep Neural Networks for Chronological Age Estimation from OPG images*. IEEE Transactions on Medical Imaging (2020).

[16] Ken Schwaber y Jeff. Sutherland, *La Guía de Scrum: Las Reglas del Juego*. (2013)

[17] Guido van Rossum, Barry Warsaw, Nick Coghlan. *PEP 8 – Style Guide for Python Code*.

[18] Guía Salarial del Sector TIC en Galicia 2019-2020: `https://www.scribd.com/document/288511179/Guia-Salarial-Sector-TI-Galicia-2015-2016`.
Consultado el 15 de junio de 2020.

[19] Teoría sobre el plano y coordenadas homogéneas. `https://es.qwe.wiki/wiki/Homogeneous_coordinates`
Consultado el 15 de marzo de 2020.

[20] Nicolás Aguirre Dobernack, Teoría de procesamiento de imágenes digital. (`http://bibing.us.es/proyectos/abreproy/12112/fichero/Documento_por_capitulos%252F3_Cap%C3%ADtulo_3.pdf`).
Consultado el 1 de junio de 2020.

[21] Definición del ruido moteado. Artículo de la wikipedia `https://en.wikipedia.org/wiki/Speckle_(interference)`
Consultado el 23 de marzo de 2020.

[22] Definición del ruido gausiano. Artículo de la wikipedia `https://en.wikipedia.org/wiki/Gaussian_noise`
Consultado el 23 de marzo de 2020.

[23] Definición del ruido *poisson*. Artículo de la wikipedia `https://es.wikipedia.org/wiki/Distribuci%C3%B3n_de_Poisson`
Consultado el 23 de marzo de 2020.

[24] Imagen OPG. `https://commons.wikimedia.org/wiki/File:OPG_w56J_-.jpg`

[25] Imagen Scrum. https://www.sinnaps.com/blog-gestion-proyectos/metodologia-scrum

[26] Imagen datos conjuntos. https://img.pixers.pics/12/02/2020

[27] Imagen coche. https://movilidadelectrica.com