



UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática



Herramienta de ayuda para la automatización de la digitalización de cursos formativos (ppt2php)

Alumno

Álvaro Saúl Rodríguez Alemán

Tutores

Dr. Alexis Quesada Arencibia

Jonathan Suárez Fuentes

Grado en Ingeniería Informática

Julio 2020 – Las Palmas de Gran Canaria

Resumen

La empresa "Fábrica de Inconformistas" se dedica parcialmente a digitalizar cursos formativos para transformarlos a cursos online partiendo de manuales en formato PowerPoint. Los cursos se componen de una serie de diapositivas, correspondiendo su apariencia visual a diversas plantillas desarrolladas por la empresa.

Este proceso resulta bastante mecánico, así que se propone la creación de una aplicación que, usando una red neuronal profunda, sea capaz de identificar a qué plantilla corresponde cada diapositiva y genere el código apropiado.

Además, la aplicación será capaz de automatizar numerosas otras tareas como la extracción de textos e imágenes de los manuales, para convertirse en el compañero ideal del equipo de digitalización, con la enorme mejora de productividad que conlleva.

Abstract

The company "Fábrica de Inconformistas" is partially dedicated to digitizing formative courses to transform them into online courses based on manuals in PowerPoint format. The courses are made up of a series of slides, and their appearance correspond to various templates developed by the company.

This process is quite mechanical, so we propose the creation of an application that, using a deep neural network, is able to identify which template each slide corresponds to and generate the appropriate code.

In addition, the application will be able to automate numerous other tasks such as the extraction of texts and images from the manuals, to become the ideal companion for the digitization team, with the enormous improvement in productivity that this entails.

Índice

Resumen.....	1
Abstract	1
Índice de ilustraciones.....	4
Índice de tablas	6
Introducción	7
Estructura del documento	9
1. Capítulo 1: Estado actual y objetivos	11
1.1. Estado actual	11
1.2. Objetivos	11
1.3. Estado del arte	12
2. Capítulo 2: Competencias específicas cubiertas	15
2.1. CP04.....	15
2.2. CP07.....	15
3. Capítulo 3: Aportaciones.....	15
4. Capítulo 4: Metodología de trabajo y planificación del proyecto.....	16
4.1. Metodología de trabajo	16
4.2. Planificación inicial del proyecto.....	16
4.3. Desviaciones sobre la planificación inicial	17
5. Capítulo 5: Tecnologías y herramientas utilizadas.....	18
5.1. Tecnologías.....	18
5.2. Herramientas.....	19
5.3. Marco teórico de las redes neuronales.....	20
5.3.1. La neurona artificial.....	20
5.3.2. La red neuronal	25
5.3.3. La red neuronal convolucional	27
6. Capítulo 6: Licencias del Software utilizado.....	30
6.1. Licencia MIT.....	30
6.2. Licencia Apache 2.0.....	30
6.3. Licencia GPL.....	31
7. Capítulo 7: Análisis	31
7.1. Actores	31
7.1.1. Programador informático.....	31
7.2. Casos de uso.....	31

7.2.1.	Diagrama de casos de uso	31
7.2.2.	Especificación de casos de uso	32
8.	Capítulo 8: Diseño	34
8.1.	Diseño de la arquitectura de la red neuronal convolucional	34
8.2.	Diseño de los conjuntos de datos usados para el entrenamiento de la CNN	36
8.3.	Diseño de la interfaz gráfica de usuario	37
9.	Capítulo 9: Desarrollo.....	39
9.1.	Estructura del proyecto.....	39
9.2.	Conjuntos de datos de imágenes	39
9.3.	Entrenamiento de la red neuronal convolucional	40
9.4.	Desarrollo de la aplicación final	47
9.4.1.	Estructura de la aplicación final	47
9.4.2.	Procesamiento y clasificación de las imágenes.....	48
9.4.3.	Generación del código usando expresiones regulares.....	49
9.4.4.	Extracción y procesamiento de las imágenes del manual.....	51
9.4.5.	Extracción y procesamiento de los textos del manual.....	53
9.4.6.	Generación de los ficheros HTML para la creación de SCORM.....	54
9.4.7.	Desarrollo de la interfaz de usuario	55
10.	Capítulo 10: Pruebas	57
10.1.	Pruebas del modelo antiguo	57
10.2.	Pruebas del modelo GDT.....	64
11.	Capítulo 11: Resultados, conclusiones y trabajo futuro.....	67
11.1.	Resultados y conclusiones.....	67
11.2.	Trabajo futuro	69
12.	Capítulo 12: Bibliografía	69
Anexo: Manual de usuario		71
<i>Convertir un manual a código</i>		71
<i>Generar los archivos HTML para generar el SCORM.....</i>		72

Índice de ilustraciones

Ilustración 1: El portal de formación smartly.es, donde se alojan todos los cursos desarrollados por la empresa	7
Ilustración 2: Diagrama de flujo del proceso de digitalización de un curso.....	12
Ilustración 3: Ejemplo de comentario de una diapositiva en Microsoft Powerpoint	13
Ilustración 4: Editor de macros integrado en Microsoft PowerPoint	14
Ilustración 5: Un Jupyter Notebook con código de muestra (Toro, s.f.)	19
Ilustración 6: Interfaz de StarUML (StarUML 3, s.f.)	20
Ilustración 7: Diagrama de un perceptrón con cinco señales de entrada (Cartas, 2020).	21
Ilustración 8: Representación de la clasificación lineal en un espacio de dos dimensiones (Guerra, El perceptrón, s.f.).....	22
Ilustración 9: Dos conjuntos de datos no separables linealmente (Guerra, El perceptrón, s.f.)	22
Ilustración 10: Expresión básica del perceptrón (Guerra, El perceptrón, s.f.)	23
Ilustración 11: Función sigmoide	23
Ilustración 12: Unidad lineal rectificada (ReLU).....	23
Ilustración 13: Representaciones gráficas de las funciones sigmoide y ReLU (Sharma, 2017)...	24
Ilustración 14: Ejemplo de función de error (Guerra, El perceptrón, s.f.)	24
Ilustración 15: Función del perceptrón para una muestra (Guerra, El perceptrón, s.f.)	25
Ilustración 16: Función de error teniendo en cuenta la función de activación (Guerra, El perceptrón, s.f.).....	25
Ilustración 17: Actualización de pesos tras cada iteración del entrenamiento (Guerra, El perceptrón, s.f.).....	25
Ilustración 18: Ejemplo de una red neuronal solucionando un problema de clasificación no separable linealmente (Guerra, Redes neuronales 1, s.f.).....	26
Ilustración 19: Propagación hacia adelante y hacia atrás en una red neuronal (Moawad, 2018)	27
Ilustración 20: Como vemos nosotros las imágenes frente a como las ven los ordenadores (Chatterjee, 2019)	28
Ilustración 21: Estructura típica de una red neuronal convolucional (Chatterjee, 2019).....	28
Ilustración 22: Visualización de un filtro de 7x7 (Chatterjee, 2019)	29
Ilustración 23: Ejemplo del funcionamiento de un filtro (Chatterjee, 2019)	29
Ilustración 24: Operación de "pooling" (Chatterjee, 2019)	30
Ilustración 25: Casos de uso de la aplicación	32
Ilustración 26: Mockup del caso de uso "Generar curso"	37
Ilustración 27: Mockup del caso de uso "Generar HTML para Scorm"	38
Ilustración 28: Script bash para renombrar todos los ficheros de una carpeta.....	39
Ilustración 29: Proceso iterativo del Machine Learning (Dan Clark, 2018).....	42
Ilustración 30: Precisión de entrenamiento del modelo antiguo	45
Ilustración 31: Evolución del "loss" del modelo antiguo.....	45
Ilustración 32: Precisión de entrenamiento del modelo GDT	46
Ilustración 33: Evolución del "loss" del modelo GDT	46
Ilustración 34: Contenido del script nun_windows.bat	47
Ilustración 35: Algoritmo de lectura de imágenes	48
Ilustración 36: Algoritmo de clasificación de las imágenes.....	49
Ilustración 37: La plantilla "126" ocupa varias diapositivas en los manuales, pero se puede representar en el código en una sola instancia de la plantilla	50

Ilustración 38: Algoritmo de extracción del código de las plantillas.....	50
Ilustración 39: Estructura de las plantillas en el repositorio.....	51
Ilustración 40: Iterador de imágenes en cada diapositiva	51
Ilustración 41: Algoritmo de extracción de las imágenes	52
Ilustración 42: Método para la inserción de las rutas de las imágenes en el código generado .	53
Ilustración 43: Algoritmo de extracción de los textos.....	54
Ilustración 44: Métodos relacionados a la generación de los ficheros HTML para la conversión de los cursos a formato SCORM.....	55
Ilustración 45: Interfaz gráfica del caso de uso "Generar curso"	56
Ilustración 46: Interfaz gráfica del caso de uso "Generar HTML para SCORM"	56
Ilustración 47: Ventana emergente con un mensaje de error al introducir las rutas incorrectamente durante la generación de un curso	57
Ilustración 48: Ventana emergente con un mensaje de éxito al completar la operación	57
Ilustración 49: Recuento de las predicciones realizadas en el curso perteneciente al modelo antiguo	63
Ilustración 50: Distribución de los errores realizados en las clasificaciones realizadas en el curso perteneciente al modelo antiguo.....	63
Ilustración 51: Recuento de las predicciones realizadas en el curso perteneciente al modelo GDT.....	66
Ilustración 52: Distribución de los errores realizados en las clasificaciones realizadas en el curso perteneciente al modelo GDT	67

Índice de tablas

Tabla 1: Planificación inicial del proyecto	17
Tabla 2: Especificación del caso de uso "Generar curso"	33
Tabla 3: Especificación del caso de uso "Generar HTML para Scorm"	34
Tabla 4: Resultados de las clasificaciones por parte de la red neuronal frente a las plantillas reales, con el curso "Oficina sin papeles - La facturación electrónica", perteneciente al modelo antiguo	62
Tabla 5: Resultados de las clasificaciones por parte de la red neuronal frente a las plantillas reales, con el curso "Capacidad de organización y organización", perteneciente al modelo GDT	66

Introducción

La empresa en la que realizamos nuestras prácticas de empresa y en la que trabajamos actualmente, “Fábrica de Inconformistas” se dedica a múltiples actividades relacionadas con la formación, tanto presencial como online. Se trata de una empresa pequeña, con un equipo de aproximadamente diez personas, entre los cuales se encuentra un equipo reducido de programadores informáticos que, además de dedicarse a algunos proyectos encargados por clientes externos, también desarrollan los cursos online de la empresa a través de la digitalización de los manuales de los mismos, que no son más que presentaciones en formato PowerPoint que especifican como debe ser la estructura visual de cada diapositiva en el curso online.

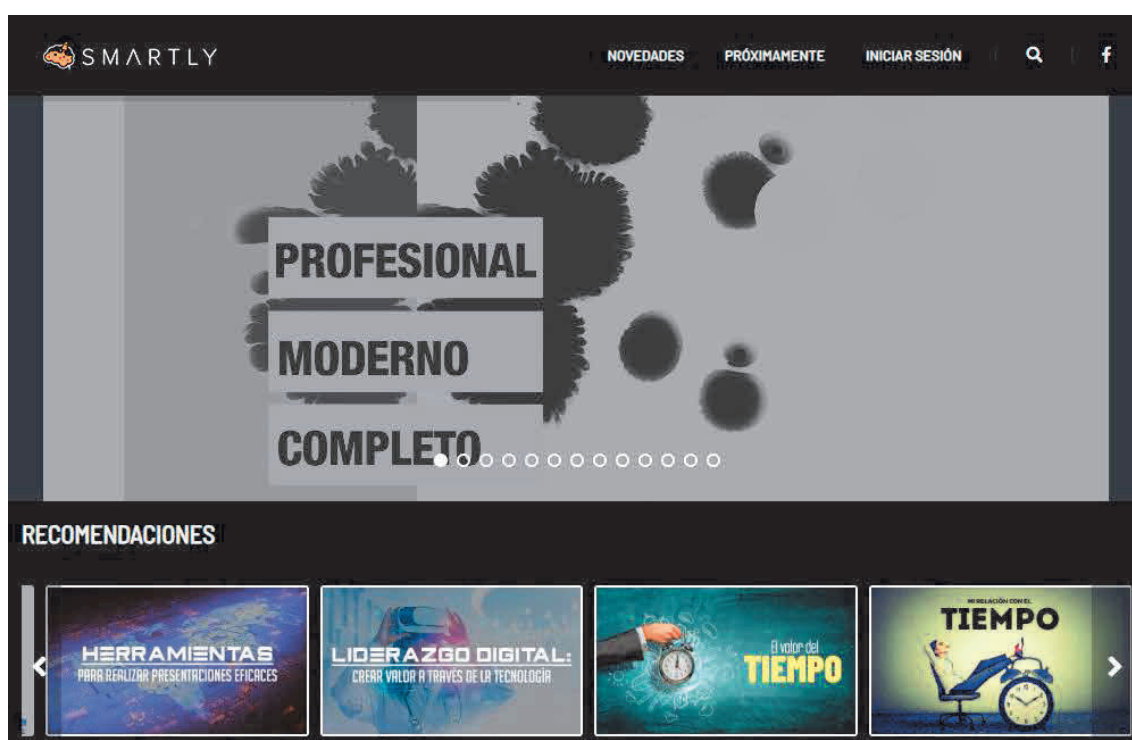


Ilustración 1: El portal de formación smartly.es, donde se alojan todos los cursos desarrollados por la empresa

Actualmente, la empresa está embarcada posiblemente en su proyecto más ambicioso desde sus comienzos: la creación de un portal de formación online llamado smartly.es (Ilustración 1), donde se alojarán todos los cursos que desarrollan y será útil tanto para empresas que deseen gestionar toda la información relacionada con la formación de sus empleados, como para particulares que deseen realizar un catálogo creciente de cursos y obtener diplomas que puedan impulsar sus currículos. El autor del presente trabajo se encuentra actualmente a cargo del desarrollo de dicho portal, junto con otros compañeros de la empresa, y ya tenemos una alianza con la Universidad de Las Palmas de Gran Canaria para su uso de cara a complementar la finalización de las prácticas de empresa de los alumnos que no han podido finalizarlas debido a la compleja situación laboral y estudiantil dada por la pandemia de COVID-19.

A medida que el enfoque del equipo de desarrollo se va trasladando más hacia el portal de formación, resulta cada vez más importante encontrar maneras de aumentar el ritmo de desarrollo de cursos para poblarlo de estos. Una solución sería contratar más personal que se dedicase a esta tarea, pero teniendo en cuenta que la digitalización de cursos es en gran medida un proceso mecánico, a través de este proyecto planteamos el desarrollo de una herramienta que pudiese realizar gran parte de este proceso de forma automática.

Así pues, tareas como detectar que plantillas de nuestro repositorio interno se están usando para representar el manual de un curso a través de una red neuronal convolucional que diseñamos y entrenamos nosotros, y generar el código del curso usándolas apropiadamente, simplificarían en gran medida el trabajo requerido por los programadores de cara a digitalización de los cursos. De esta forma, sus tareas se verían reducidas a extraer los textos y las imágenes de las diapositivas del manual e introducirlos en el código, pero hasta en eso la herramienta les asistiría, realizando dichas extracciones de forma automática, insertando las etiquetas HTML necesarias en los textos y situando las rutas de las imágenes en las variables correctas en el código si siguen la nomenclatura estándar.

La herramienta desarrollada en este proyecto se planteará como la navaja suiza de los integrantes del equipo de digitalización de la empresa, que asistirá, simplificará y agilizará el trabajo de estos, aumentando en gran medida su rendimiento de cara a la producción de los cursos.

Estructura del documento

La estructura de este documento se basa en varios capítulos, cuyos propósitos explicamos a continuación:

- Capítulo 1: Estado actual y objetivos.

En este capítulo mostramos el estado actual del problema, además de los objetivos que planteamos con el proyecto a desarrollar para solucionarlo y el estado del arte previo al desarrollo de este.

- Capítulo 2: Competencias específicas cubiertas.

En este capítulo listamos las competencias específicas cubiertas en este proyecto junto a las justificaciones pertinentes.

- Capítulo 3: Aportaciones.

En este capítulo explicamos las aportaciones que tiene este proyecto en el entorno empresarial, económico y personal.

- Capítulo 4: Metodología de trabajo y planificación del proyecto.

En este capítulo tratamos la metodología de trabajo que hemos seguido para desarrollar el proyecto, además de la planificación temporal. Además, mostraremos las desviaciones sobre la planificación inicial que se dieron una vez finalizado el desarrollo.

- Capítulo 5: Tecnologías y herramientas utilizadas.

En este capítulo listamos las tecnologías y herramientas sobre las que nos hemos apoyado para desarrollar el proyecto, explicándolas brevemente. Se incluye una explicación del marco teórico de las redes neuronales, la tecnología que hace que este proyecto sea posible.

- Capítulo 6: Licencias del software utilizado.

En este capítulo listamos las licencias bajo las que están el software de terceros que usamos en el proyecto explicamos sus características y las limitaciones que presentan.

- Capítulo 7: Análisis.

En este capítulo explicamos los actores que usarán la herramienta, además de los casos de uso posibles y sus especificaciones completas.

- Capítulo 8: Diseño.

En este capítulo explicamos aspectos fundamentales del diseño de la arquitectura de la red neuronal que hemos usado, de los conjuntos de datos confeccionados y de la interfaz de usuario de la aplicación final.

- Capítulo 9: Desarrollo.

En este capítulo explicaremos las distintas fases por las que pasamos durante el desarrollo de la aplicación, incluyendo los fragmentos de código más relevantes.

- Capítulo 10: Pruebas.

En este capítulo analizaremos y compartiremos los resultados del procesamiento de dos cursos distintos usando la herramienta, cada uno de ellos basado en un modelo de plantillas distinto.

- Capítulo 11: Resultados, conclusiones y trabajo futuro.

En este capítulo analizaremos los resultados obtenidos y explicaremos las conclusiones a las que hemos llegado a través del desarrollo del proyecto. Además, listaremos una serie de funcionalidades adicionales y mejoras que se le podrían realizar a la aplicación en el futuro que mejorarían la experiencia de uso.

- Capítulo 12: Bibliografía.

En este capítulo listamos todas las referencias bibliográficas usadas a lo largo de esta memoria.

- Anexo: Manual de usuario.

En el anexo incluimos un breve manual de uso de la herramienta, enfocada a los programadores de la empresa.

1. Capítulo 1: Estado actual y objetivos

1.1. Estado actual

La empresa “Fábrica de Inconformistas”, se dedica, entre otras actividades, a digitalizar cursos formativos para transformarlos a cursos online dinámicos e interactivos partiendo de manuales estáticos en formato PowerPoint. Los cursos son en esencia páginas web, y se componen de una serie de diapositivas divididas en módulos, correspondiendo el apartado visual de cada diapositiva a diversas plantillas compuestas por código PHP, JavaScript, CSS y HTML que han sido desarrolladas por el equipo de ingenieros de la empresa. Además, los cursos disponen de secciones adicionales como una Guía Didáctica, una Bibliografía, un Glosario, y demás apartados que facilitan el aprendizaje para los usuarios.

El proceso de digitalización de los cursos resulta un tanto mecánico y tedioso. Por cada una de las diapositivas del manual (un fichero PowerPoint), hay que buscar en el repositorio de plantillas desarrolladas una en la que se pueda representar apropiadamente la diapositiva, copiar el código y pegarlo en un fichero en el que se detalla la sucesión de diapositivas del curso, y finalmente configurar los parámetros de la plantilla, entre los que se incluyen parámetros visuales, textos y rutas a imágenes, y así sucesivamente hasta completar los cientos de diapositivas que pueden componer un manual. Actualmente, el repositorio alberga cerca de 100 plantillas distintas.

1.2. Objetivos

Se propone la creación de una aplicación que permita la automatización del proceso de digitalización de cursos formativos online. Entre los objetivos del trabajo se encuentran los siguientes:

- Determinar qué plantilla corresponde a cada diapositiva con un mínimo de 90% de precisión, y generar los ficheros correspondientes a cada uno de los módulos, con los códigos correspondientes a cada una de las plantillas detectadas.
- Extraer las imágenes y textos usados en el manual e insertarlos en el código generado en lugares apropiados.
- Desarrollar una interfaz gráfica para la aplicación que sea intuitiva, informativa y con una curva de aprendizaje lo más accesible posible.
- Operar con los distintos modelos de estilo que tiene cada uno de los cursos (los cursos que pertenecen a distintos programas formativos tienen apartados estilísticos muy variantes y usan otras plantillas).
- Recordar las preferencias de configuración de los usuarios para evitar que tengan que introducirlas en cada uso.

- Desarrollar utilidades extra que automaticen otros apartados del proceso de digitalización, como la generación de los ficheros HTML necesarios para convertir el curso a formato SCORM.

1.3. Estado del arte

Antes de tratar el estado del arte, conviene que profundicemos más en explicar el proceso de digitalización de cursos con la ayuda del siguiente diagrama de flujo:

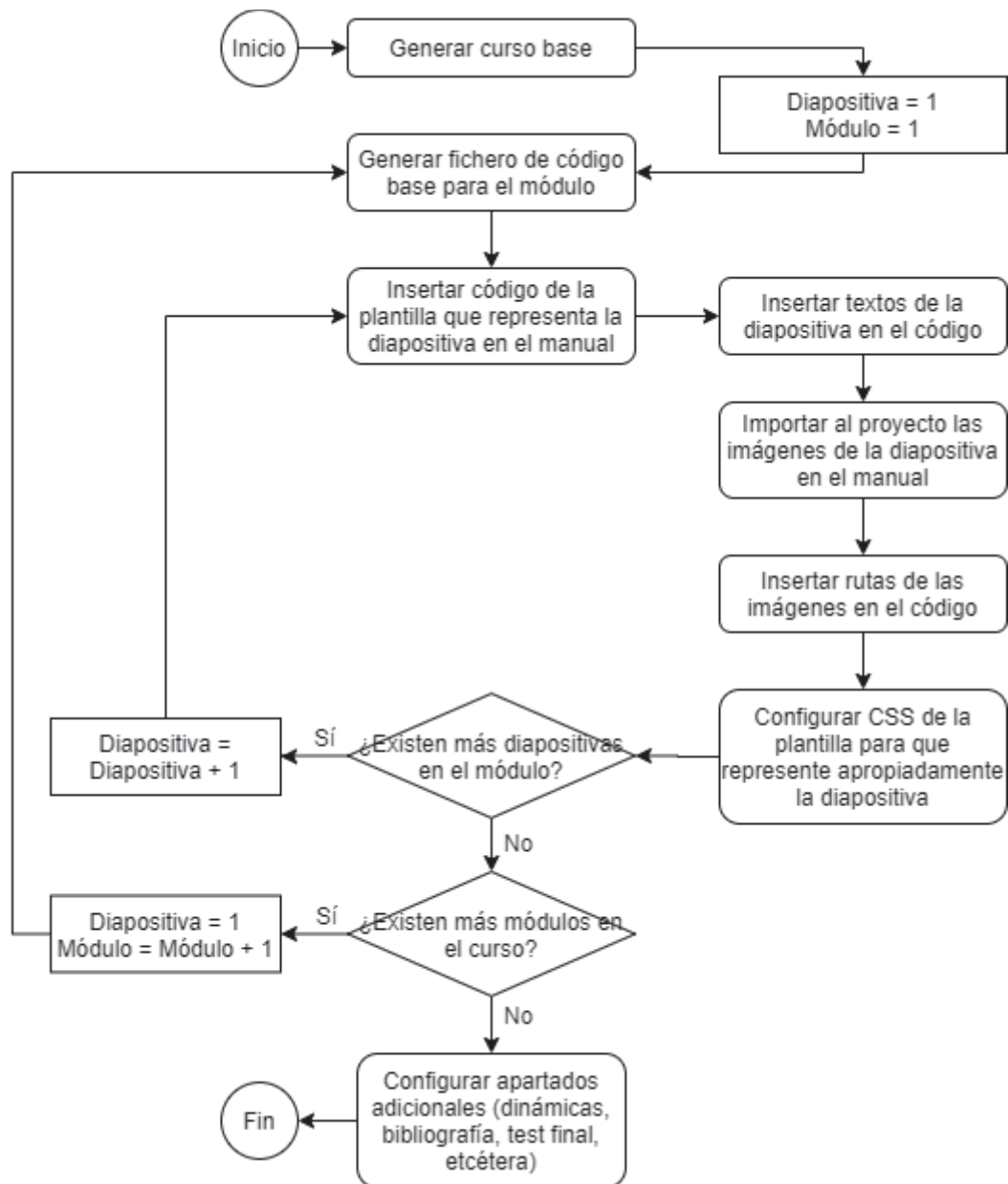


Ilustración 2: Diagrama de flujo del proceso de digitalización de un curso

Así pues, como podemos observar en la Ilustración 2, la parte más tediosa de la digitalización de cursos resulta ser la configuración de cada diapositiva del manual, obligando al empleado a identificar las plantillas usadas en cada diapositiva del manual, acudir constantemente al

repositorio de plantillas para obtener el código correspondiente, insertar los textos y las imágenes en el código y por último añadir las cláusulas CSS necesarias para que se represente correctamente el apartado visual de la misma.

La generación de los ficheros base resulta trivial, ya que disponemos de ejemplos de estos en el repositorio, que duplicamos y adaptamos a nuestras necesidades en un tiempo muy reducido. Por otra parte, la creación de los apartados adicionales sí que conlleva más tiempo de trabajo, pero sigue siendo mucho menor que la configuración de las diapositivas y sin duda mucho menos monótono.

Debido a lo particular del problema que exponemos y lo adaptado que está a la realidad concreta de la empresa resulta difícil encontrar alternativas existentes en el mercado actual que lo solucionen. Así pues, más que alternativas a nuestro proyecto en el mercado, lo más que podemos encontrar son formas distintas de desarrollar una solución para los problemas expuestos, que no resultarían en un aumento del rendimiento tan grande o una solución tan universal a dichos problemas.

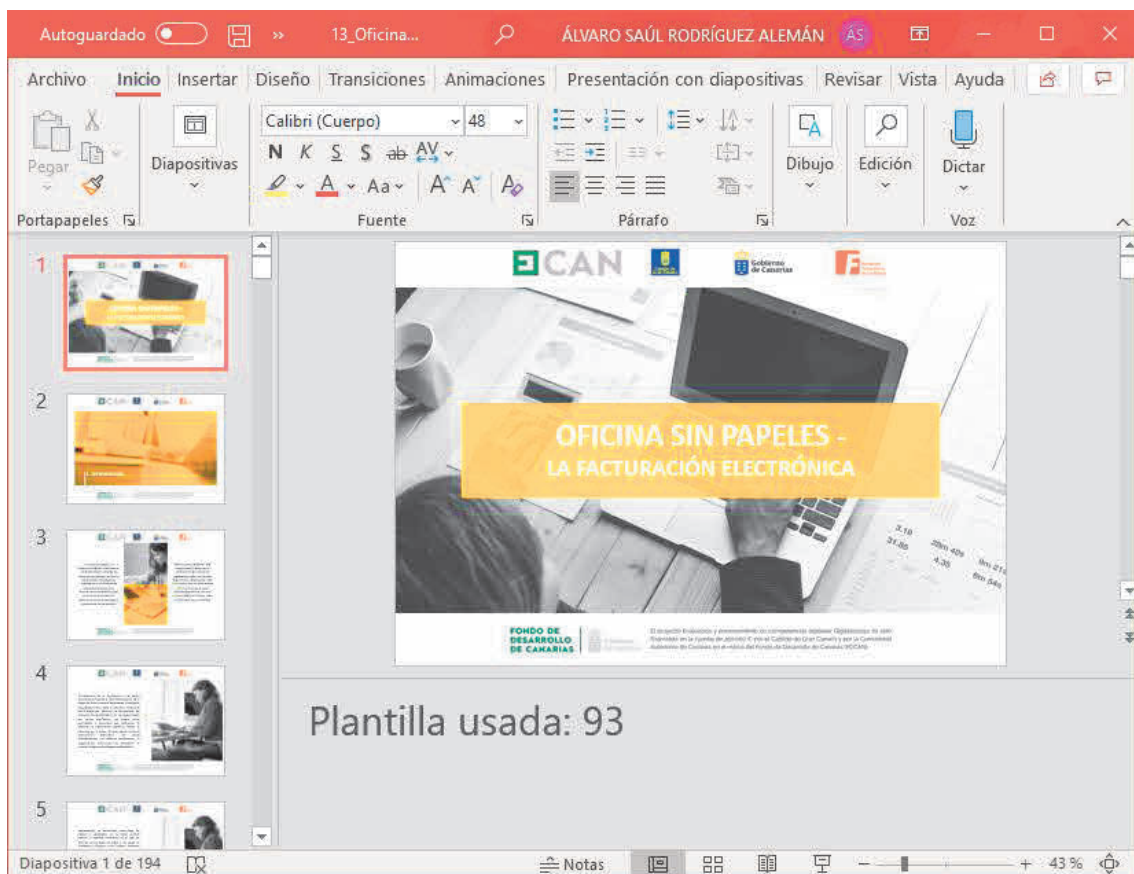


Ilustración 3: Ejemplo de comentario de una diapositiva en Microsoft Powerpoint

Por ejemplo, una posible solución al problema podría ser el hacer que el equipo que desarrolla los manuales insertase comentarios en cada una de las diapositivas de estos indicando el código de la plantilla usada, como podemos observar en la Ilustración 3.

De esta forma, se podría desarrollar un software reducido que simplemente se limitase a la generación del código usando esta información. Sin embargo, de esta manera estamos obligando a dicho equipo a realizar trabajo adicional, insertando en cada uno de los cientos de

diapositivas que hay en los manuales un comentario indicando información que no necesariamente deben tener (el equipo de desarrollo de manuales no diseña el estilo de las diapositivas a partir de las plantillas, sino que las plantillas se diseñan a posteriori para representar los estilos que dicho equipo realiza).

Además, esta medida no sería automáticamente retroactiva: los manuales desarrollados previamente a la implementación de esta alternativa no se acogerían a la misma a no ser que alguien le dedicase tiempo a ir clasificando manualmente cada una de las diapositivas e insertando los comentarios acordes.

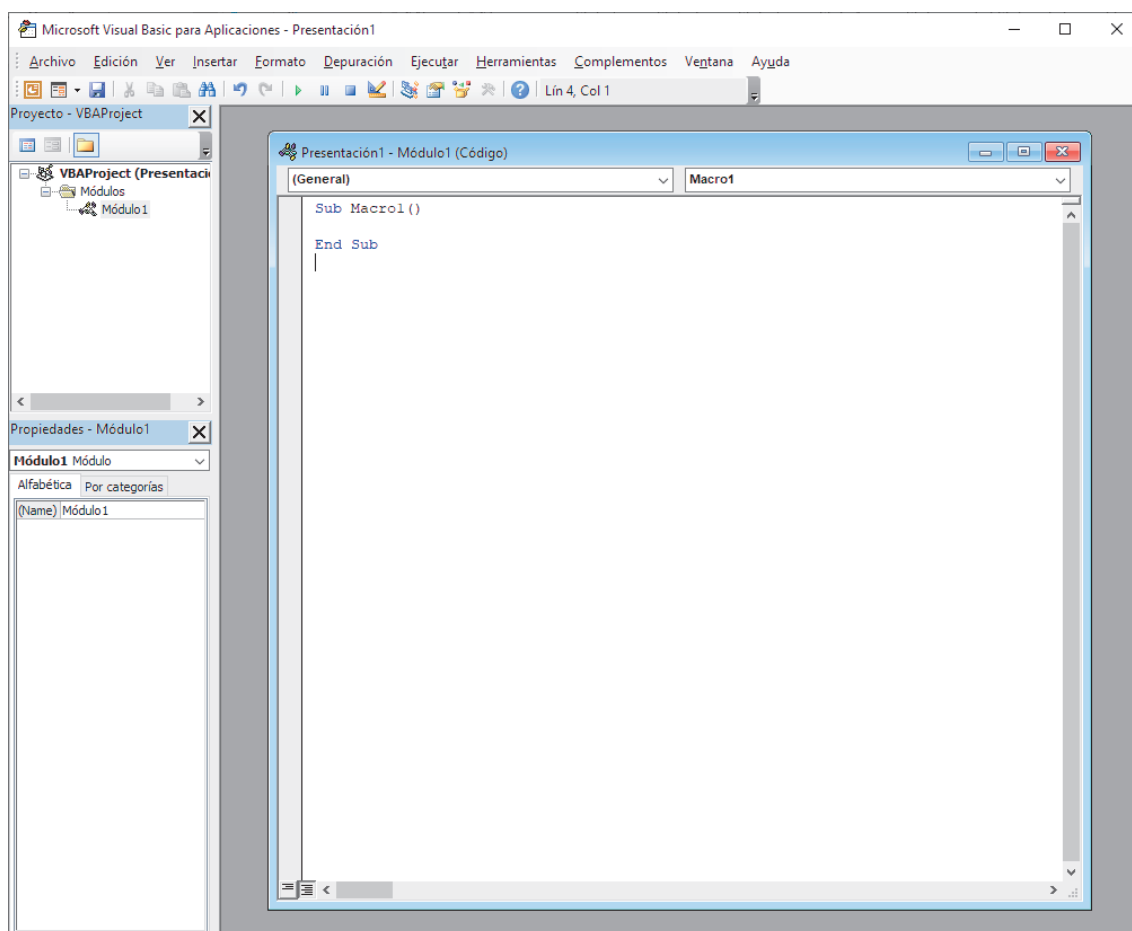


Ilustración 4: Editor de macros integrado en Microsoft PowerPoint

Por otra parte, de cara a la extracción de los textos e imágenes de los manuales, se podrían desarrollar macros VBA que se ejecutaran en los propios manuales que hiciesen esta tarea. Sin embargo, resulta más tedioso trabajar usando esta tecnología, ya que dispone de una documentación y un soporte comunitario bastante reducido. Además, las herramientas integradas disponibles para trabajar en las macros, como la que podemos ver en la Ilustración 4, son mucho menos completas que los entornos de desarrollo tradicionales.

En definitiva, el uso de esta alternativa podría funcionar, pero sin duda desembocaría en que sería mucho más laborioso mantener el código y mucho más difícil para otros programadores ajenos al proyecto de cara a incorporarse e implementar funcionalidades nuevas.

2. Capítulo 2: Competencias específicas cubiertas

2.1. CP04

“Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación”

En este trabajo de fin de título se usan los conocimientos sobre los fundamentos propios de los sistemas inteligentes, concretamente las redes neuronales convolucionales, para construir una aplicación informática basada en las mismas.

2.2. CP07

“Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.”

En este proyecto realizamos el diseño y el entrenamiento de una red neuronal convolucional con un conjunto de datos compuesto por un gran número de imágenes, y dicha red se usa en la aplicación final desarrollada.

3. Capítulo 3: Aportaciones

Teniendo en cuenta el carácter mecánico del proceso de trabajo, una herramienta que detecte a qué plantilla corresponde cada diapositiva e inserte el código correspondiente en el fichero de trabajo redundaría en un aumento significativo del rendimiento del equipo de digitalización, reduciendo su coste y por consiguiente aumentando los beneficios de la empresa relacionados con dicha actividad.

Además, el aumento de la velocidad de producción de cursos es vital para el éxito del portal smartly.es, de cara al lanzamiento constante de nuevos cursos que resulten atractivos a nuevos clientes. Teniendo en cuenta que el enfoque principal de la actividad de la empresa es ahora el desarrollo de dicho portal, esta herramienta resulta ser de vital importancia.

La herramienta no busca reemplazar la labor humana, sino agilizarla, enriquecerla y reducir el conocimiento técnico necesario para llevar a cabo la digitalización. Los ficheros generados disponen de las plantillas correctas en el orden correcto, pero deberán ser configuradas posteriormente con los textos y las imágenes que deberán mostrar, o las directivas CSS necesarias para su correcta visualización. Sin embargo, esta tarea apenas ocupa tiempo e incluso en algunos casos la propia herramienta lo hace automáticamente, como con las

imágenes, en el caso de que la plantilla use la nomenclatura estándar en las variables que contienen las rutas (lo detallaremos en el apartado de desarrollo).

4. Capítulo 4: Metodología de trabajo y planificación del proyecto

4.1. Metodología de trabajo

La metodología que se ha seguido en el trabajo consistirá en la sucesión de iteraciones con un número de funcionalidades incrementales. Las iteraciones y las funcionalidades que se verán incluidas en las mismas seguirán los objetivos establecidos, y además atenderán a las necesidades y la retroalimentación del equipo de desarrollo al usar la herramienta durante su desarrollo.

Sin embargo, antes de desarrollar la herramienta en sí, hubo que dedicarle tiempo a la recolección de imágenes para el conjunto de datos que requiere la red neuronal, así como para confeccionar y entrenar a la misma.

Así pues, tras realizar las tareas preliminares mencionadas, las iteraciones de desarrollo de la aplicación final fueron las siguientes:

- **Iteración 1:** Procesamiento de las imágenes y clasificación de estas.
- **Iteración 2:** Generación del código usando expresiones regulares.
- **Iteración 3:** Extracción y procesamiento de las imágenes del manual.
- **Iteración 4:** Extracción y procesamiento de los textos del manual.
- **Iteración 5:** Generación de los ficheros HTML para la creación de SCORM.
- **Iteración 6:** Desarrollo de la interfaz de usuario.

4.2. Planificación inicial del proyecto

<i>Fases</i>	<i>Duración estimada (horas)</i>	<i>Tareas</i>
<i>Estudio previo / Análisis</i>	20	<p>Tarea 1.1: Estudio previo de las funcionalidades que formarán parte del proyecto final y sus particularidades.</p> <p>Tarea 1.2: Estudio previo de la arquitectura de la red neuronal que se usará.</p> <p>Tarea 1.3: Análisis de la estructura que tendrá la interfaz gráfica del proyecto.</p>

<i>Diseño / Desarrollo / Implementación</i>	180	<p>Tarea 2.1: Recopilación de la base de datos de imágenes de diapositivas que corresponden a cada plantilla.</p> <p>Tarea 2.2: Diseño y entrenamiento de la red neuronal.</p> <p>Tarea 2.3: Desarrollo de la aplicación final.</p>
<i>Evaluación / Validación / Prueba</i>	20	<p>Tarea 3.1: Validación del funcionamiento de la aplicación y evaluación de la mejora del rendimiento.</p>
<i>Documentación / Presentación</i>	80	<p>Tarea 4.1: Creación de la documentación del proceso de desarrollo.</p> <p>Tarea 4.2: Estudio de la estructura de la presentación del proyecto.</p>

Tabla 1: Planificación inicial del proyecto

Así pues, como podemos observar en la Tabla 1, la mayor carga de trabajo en términos de duración reside en el diseño, desarrollo e implementación, y resulta lógico por el carácter práctico del proyecto. Además, debido a lo sencilla que es la idea y los pocos casos de uso de esta, el diseño tampoco ocupará muchas de las 180 horas asignadas, dejándolas principalmente para las fases de desarrollo e implementación.

Por otra parte, las fases de estudio previo y análisis, además de la evaluación, validación y pruebas disponen de apenas 20 horas, pero no por ello significa que resulten de menor importancia. Estaba previsto que el estudio previo iba a resultar ser poca carga de trabajo, porque gran parte de este ya se había realizado en el prototipo previo a este proyecto. Por otra parte, las pruebas sí que sería necesario hacerlas desde cero, pero su realización y análisis no debería durar más de las 20 horas especificadas.

4.3. Desviaciones sobre la planificación inicial

A lo largo del proyecto hemos sufrido ciertas desviaciones sobre la planificación inicial del proyecto. Para empezar, con respecto a la fase de estudio previo y análisis, estuvimos en lo cierto dedicándole una estimación de tiempo reducida ya que habíamos realizado en el prototipo previo muchas de las tareas que se realizan en esta fase. Sin embargo, incluso pese a ese hecho sobreestimamos el tiempo requerido, siendo necesarias menos de la mitad de las horas presentes en la planificación. Esto se debe a que la idea de este proyecto fue gestándose a lo largo de varios meses previamente al desarrollo de este. Conocíamos de primera mano lo que debía hacer la herramienta para que nos fuese útil, y, además, los demás compañeros de trabajo nos aportaron ideas muy interesantes como la generación de los ficheros HTML para la conversión de los cursos a SCORM.

Por otra parte, la fase de desarrollo ocupó unas horas más de las estimadas, principalmente por dos razones: por la recolección de los conjuntos de datos y por la extracción de las imágenes y textos de los manuales.

En primer lugar, subestimamos el tiempo que sería necesario para confeccionar los conjuntos de datos, ya que no tuvimos en cuenta que ir clasificando manualmente cientos de imágenes una a una podría ser tan tedioso y llevar tanto tiempo. Además, por si fuera poco, tuvimos que dedicarle tiempo a corregir los errores en dichos conjuntos de datos, errores previsibles teniendo en cuenta lo fácil que es equivocarse al clasificar algunas imágenes o ponerlas en carpetas incorrectas teniendo que etiquetar cientos de ellas en el menor tiempo posible. Con que unas pocas imágenes estén mal clasificadas puede significar una reducción significativa de la precisión de las redes, así que teníamos que ir con mucho cuidado asegurándonos de que cada imagen estaba situada donde debía.

Como hemos mencionado anteriormente, otro factor que nos costó más tiempo del estimado fue la extracción de las imágenes y los textos de los manuales. Para realizar esta tarea debíamos usar la librería Python-pptx, que dispone de documentación, pero quizás no es todo lo extensa y esclarecedora que podría ser. Además, teniendo en cuenta que los conceptos relacionados con las estructuras de datos que se manejan en los ficheros PowerPoint no son muy fáciles de comprender a primera vista, tuvimos que pasar por una fase de prueba y error ligeramente frustrante hasta que conseguimos que funcionase.

Por último, con respecto a las fases de pruebas y confección de la documentación, en este caso sí que estimamos correctamente, habiendo invertido en las mismas la cantidad de tiempo estimada sin apenas variaciones.

5. Capítulo 5: Tecnologías y herramientas utilizadas

5.1. Tecnologías

- **Python:** Lenguaje de programación interpretado muy útil para el manejo de grandes conjuntos de datos, el análisis matemático y la implementación de redes neuronales. Las razones por las que Python resulta ser una opción de lenguaje de programación idónea para tareas relacionadas con la Inteligencia Artificial y el *Machine Learning* son muchas: desde su flexibilidad y legibilidad, hasta su gran ecosistema de librerías y opciones de visualización, pasando por su creciente popularidad y excelente soporte comunitario. (Luashchuk, s.f.)
- **Git:** Software de control de versiones robusto y fácil de usar que favorece el desarrollo colaborativo. Es fácil de aprender, es de código abierto, y es capaz de manejar proyectos de todo tipo de tamaños, con funciones que otros sistemas de control de versiones más antiguos no tienen, como los flujos de trabajo múltiples.
- **Keras:** API de alto nivel para el desarrollo de redes neuronales escrita en Python y que usa *TensorFlow*, *CNTK* o *Theano* como base. Se centra en habilitar el prototipado veloz a través de su facilidad de uso y configuración, y soporta tanto redes neuronales convolucionales como recurrentes.

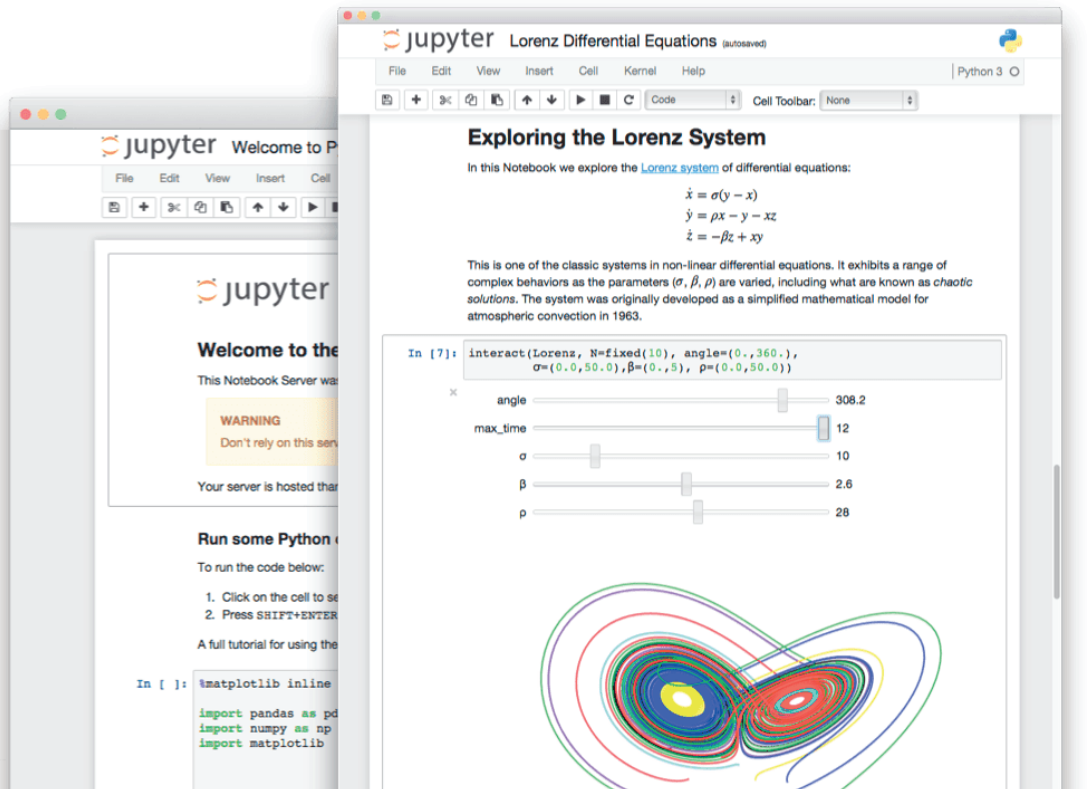


Ilustración 5: Un Jupyter Notebook con código de muestra (Toro, s.f.)

- **Jupyter Notebook:** Aplicación web de código abierto que permite crear y compartir documentos que contienen código ejecutable, además de texto con estilos y visualizaciones. El uso de esta tecnología es esencial para el modelado y entrenamiento de redes neuronales, ya que nos permite ejecutar modificar y ejecutar segmentos del código al momento, otorgando una flexibilidad enorme que no tendríamos trabajando en un IDE tradicional, como podemos observar en la Ilustración 5 (Toro, s.f.).
- **Tensorflow:** Plataforma de código abierto para *Machine Learning*. Dispone de una amplia gama de herramientas, librerías y recursos para el desarrollo de aplicaciones que usan el aprendizaje automático. Lo usamos indirectamente a través de *Keras*.

5.2. Herramientas

- **Pycharm Community Edition:** IDE gratuito desarrollado por JetBrains para el desarrollo de aplicaciones en Python. Dispone de múltiples herramientas muy útiles para el desarrollo, como finalización del código inteligente, indicaciones de errores y diversos métodos de refactorización automática.
- **Google Colab:** Entorno de desarrollo online que permite ejecutar Jupyter Notebooks de Python en los servidores de Google con tarjetas gráficas dedicadas. Muy útil para el entrenamiento de redes neuronales sin tener que alojar los Jupyter Notebooks en nuestros ordenadores ni usar nuestras propias tarjetas gráficas.

- **GitHub:** Herramienta de desarrollo para equipos que también permite el alojamiento de repositorios Git.

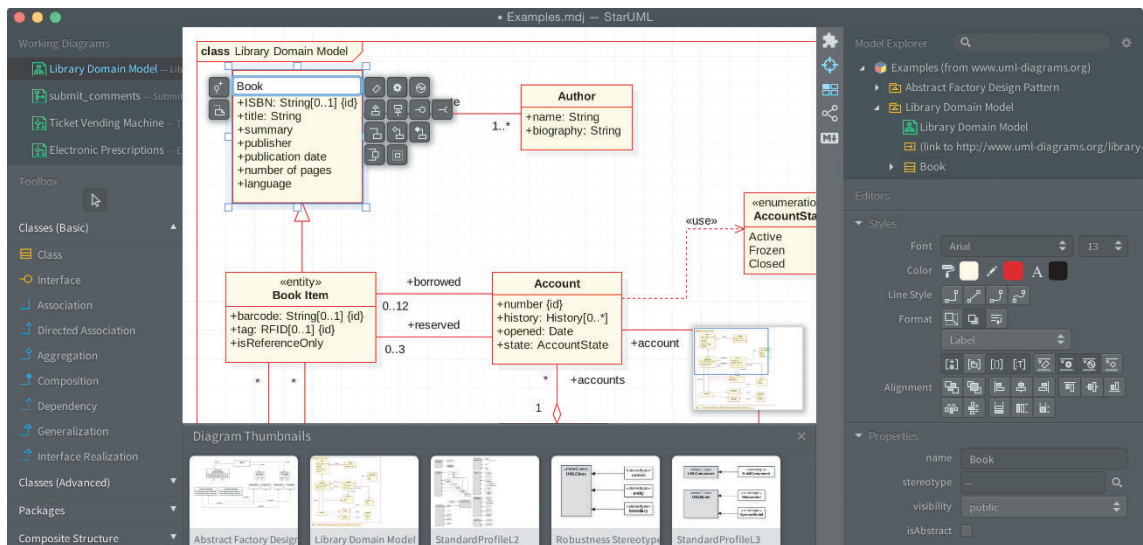


Ilustración 6: Interfaz de StarUML (StarUML 3, s.f.)

- **StarUML:** Aplicación de escritorio que permite la creación de una amplia gama de diagramas UML de forma intuitiva y sencilla, cuya interfaz podemos observar en la Ilustración 6 (StarUML 3, s.f.).
- **Google Chrome:** Navegador propio de Google. Junto con Mozilla Firefox, es el navegador web que mayor compatibilidad ostenta con los cursos online desarrollados por la empresa.
- **Google Drive:** Herramienta de almacenamiento de archivos propia de Google. Dispone de integración con los Jupyter Notebooks ejecutados a través de Google Colab, y resulta el repositorio perfecto para los conjuntos de datos que usamos para entrenar las redes neuronales.
- **Microsoft PowerPoint:** Software de presentaciones hecho por Microsoft con mayor compatibilidad con los manuales de los cursos que realiza la empresa.

5.3. Marco teórico de las redes neuronales

Debido a la enorme importancia que tienen las redes neuronales en este proyecto, dedicaremos una pequeña sección de esta memoria a explicar con cierto grado de profundidad el funcionamiento de esta tecnología. En primer lugar, explicaremos el funcionamiento de la neurona artificial de forma individual, luego el funcionamiento de las redes neuronales y, por último, el concepto de red neuronal convolucional.

5.3.1. La neurona artificial

La neurona artificial, también conocida como perceptrón, es la unidad básica de las redes neuronales y de por sí sola actúa como un clasificador lineal, de tal forma que es capaz de

clasificar un conjunto de datos entre dos clases posibles. Su funcionamiento es similar al de una neurona real: un perceptrón tiene unas entradas de datos (las dentritas de la neurona), un núcleo de procesamiento de estos (el soma o cuerpo celular) y una salida del resultado (el axón).

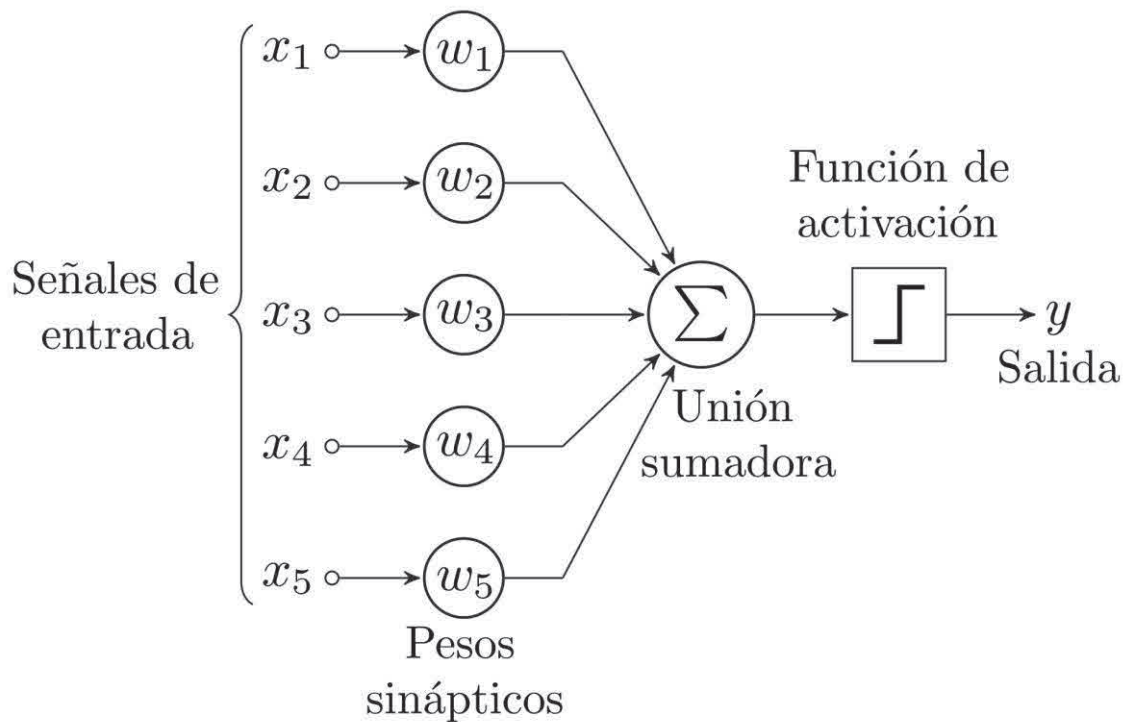


Ilustración 7: Diagrama de un perceptrón con cinco señales de entrada (Cartas, 2020).

Como podemos observar en la Ilustración 7 (Cartas, 2020), la neurona recibe un vector lineal de señales de entrada, se ponderan a través de los pesos sinápticos, se suman y procesan a través de la función de activación, y finalmente se emiten como resultado, representando la clasificación final de los datos realizada por el perceptrón. En los próximos apartados profundizaremos más sobre los conceptos o fases mencionados.

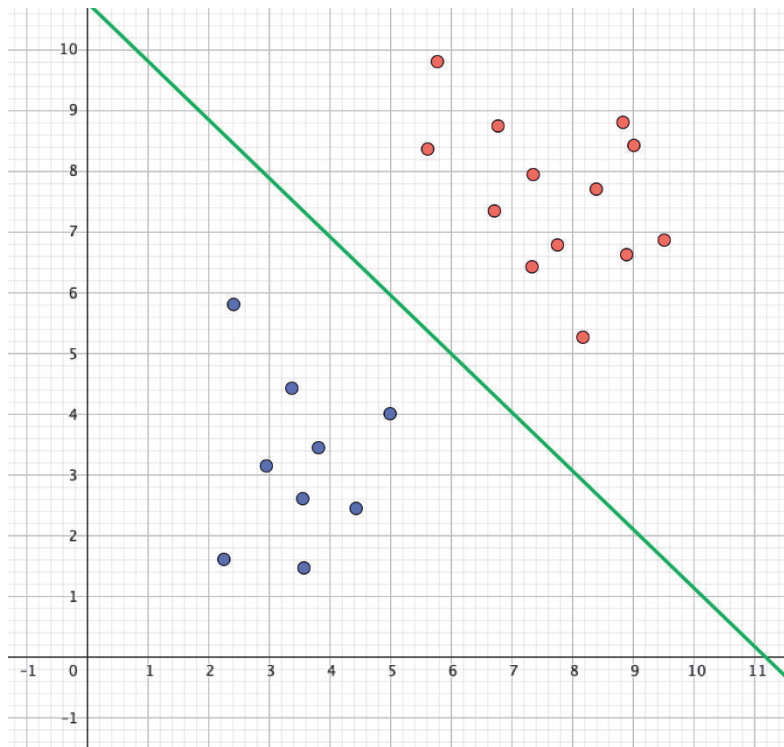


Ilustración 8: Representación de la clasificación lineal en un espacio de dos dimensiones (Guerra, El perceptrón, s.f.)

La idea del perceptrón reside en la capacidad de clasificación entre dos conjuntos de datos etiquetados apropiadamente, con la limitación de que, si representamos los datos gráficamente, debe poder existir un hiperplano capaz de escindir los conjuntos, como podemos observar en la Ilustración 8 (Guerra, El perceptrón, s.f.). En el caso de un espacio de dos dimensiones, la ecuación que define la neurona describe una recta.

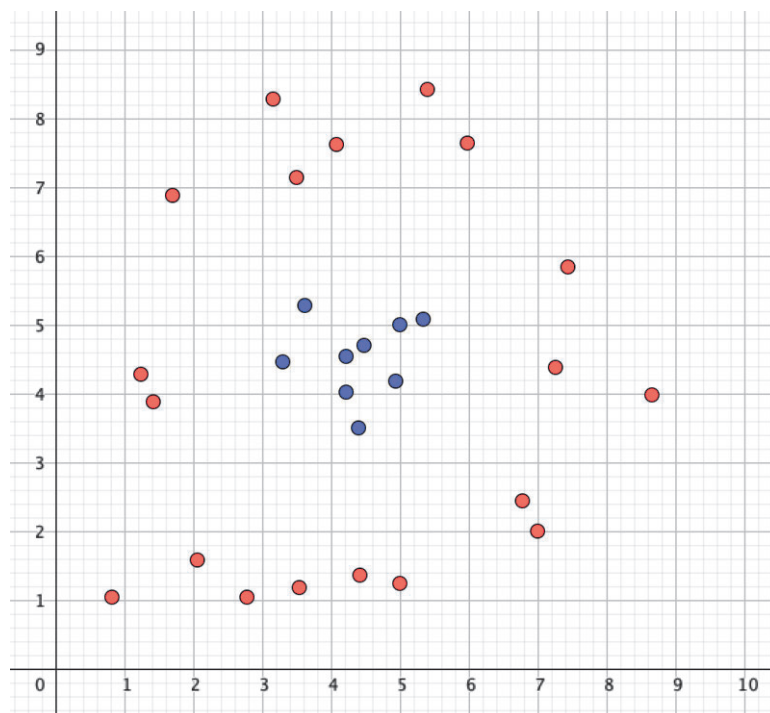


Ilustración 9: Dos conjuntos de datos no separables linealmente (Guerra, El perceptrón, s.f.)

En caso de que la clasificación entre los conjuntos requiera otra forma geométrica más compleja que un hiperplano, como podemos observar en el caso de la Ilustración 9 (Guerra, El perceptrón, s.f.), será necesario combinar varias neuronas en redes.

5.3.1.1. *Ponderación de las señales de entrada a través de los pesos sinápticos y obtención del resultado*

$$f(\mathbf{e}) = \begin{cases} 1, & \text{si } \sum_{i=1}^n w_i e_i \geq \theta \\ 0, & \text{en caso contrario} \end{cases}$$

Ilustración 10: Expresión básica del perceptrón (Guerra, El perceptrón, s.f.)

Como podemos observar en la Ilustración 10 (Guerra, El perceptrón, s.f.), en su expresión más básica la obtención del resultado de la clasificación de una entrada de datos consiste en realizar un sumatorio del producto escalar entre cada una de las señales de entrada (\mathbf{e}) y los pesos sinápticos correspondientes (\mathbf{w}), y comprobar si supera cierto umbral de activación (θ). En caso de que lo haga, el resultado es 1, y en caso negativo, el resultado es 0.

Tanto los pesos sinápticos como el umbral de activación son valores aprendidos: comienzan siendo asignados de forma aleatoria, pero como explicaremos en un apartado futuro, a medida que clasificamos datos con la neurona y la entrenamos para reducir los errores, dichos valores se van ajustando automáticamente.

5.3.1.2. *La función de activación*

En los casos reales no usamos la función de activación mostrada anteriormente por una sencilla razón: dicha función no es derivable. La derivabilidad es esencial a la hora de entrenar a la red para reducir los errores y mejorar la precisión de las clasificaciones. Además, al pasar la salida de la neurona por una función de activación nos aseguramos de que el valor se encuentra entre unas franjas establecidas (las determinadas por la función de activación), lo cual puede favorecer el entrenamiento.

$$\text{Sig}(x) = \frac{1}{1 + e^{-x}}$$

Ilustración 11: Función sigmoide

Un ejemplo de una función de activación más usada comúnmente es la función sigmoide, representada en la Ilustración 11, que devuelve un valor entre 0 y 1, con la característica de que siempre es derivable. En valores de x distintos de 0 el resultado se aproxima a 0 o a 1 dependiendo si el valor es negativo o positivo, respectivamente, y si x es 0 el resultado es 0.5.

$$f(x) = \max(0, x)$$

Ilustración 12: Unidad lineal rectificadora (ReLU)

Sin embargo, en los próximos apartados se podrá apreciar como la función de activación que más usamos en el proyecto es la función de unidad lineal rectificada (Rectified Linear Unit, ReLU), cuya definición encontramos en la Ilustración 12. Esta función es más restrictiva con los valores de x negativos, devolviendo 0 para cada uno de ellos. Esto puede significar una pérdida de capacidad de la neurona para aprender de los datos correctamente (y por ello existen alternativas como Leaky ReLU (Chris, 2019)), pero en nuestro caso ha funcionado correctamente sin causarnos ningún problema.

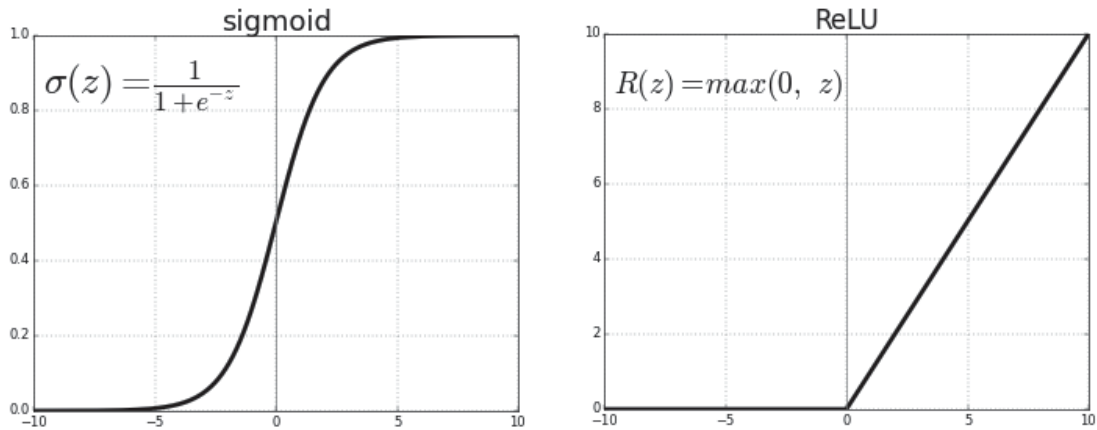


Ilustración 13: Representaciones gráficas de las funciones sigmoide y ReLU (Sharma, 2017)

En la Ilustración 13 (Sharma, 2017) podemos observar una comparativa entre las representaciones gráficas de ambas funciones de activación mencionadas. Por definición, la función ReLU no está limitada entre 0 y 1, como la función sigmoide.

5.3.1.3. La función de error

Para el entrenamiento y el consecuente aprendizaje de la neurona, es necesario explicar primero el concepto de la función de error. La función de error nos permite determinar con un valor numérico cuan equivocada ha estado la neurona al clasificar un conjunto de muestras. El objetivo del entrenamiento será reducir el resultado de dicha función, aumentando la precisión de las clasificaciones en consecuencia.

$$error = \sum_{j=1}^m (\sum_{i=0}^n w_i e_i^j - label_j)^2$$

Ilustración 14: Ejemplo de función de error (Guerra, El perceptrón, s.f.)

En la Ilustración 14 (Guerra, El perceptrón, s.f.) podemos observar un ejemplo de una función de error. En este caso, “w” hace referencia a los pesos, “e” a las señales de entrada de cada muestra y “label” a la clase (también conocida como etiqueta) a la que pertenece realmente la muestra. El sumatorio n itera sobre cada señal de entrada de una muestra, y el sumatorio m sobre todas las muestras.

Existen múltiples tipos de funciones de errores, cada una con sus ventajas y desventajas dependiendo de los detalles concretos de cada proyecto. En nuestro caso hemos usado la denominada “Categorical Cross-Entropy”, que nos ha proporcionado buenos resultados.

5.3.1.4. El aprendizaje de la neurona

Habiendo explicado todos los conceptos mencionados, ya podemos adentrarnos en el proceso de entrenamiento en sí. Para que la neurona aprenda, aplicamos un proceso denominado “descenso por el gradiente”, mediante el cual podemos ir actualizando progresivamente los pesos de nuestra neurona con el objetivo de minimizar el resultado de la función de error y mejorar la precisión.

$$h_{\vec{w}}(\vec{e}) = \text{Sig}\left(\sum_{i=0}^n w_i e_i\right)$$

Ilustración 15: Función del perceptrón para una muestra (Guerra, El perceptrón, s.f.)

En primer lugar, calculamos la función del perceptrón para cada muestra, definida como se aprecia en la Ilustración 15 (Guerra, El perceptrón, s.f.). Este ejemplo toma como función de activación la función sigmoide, pero en su lugar puede ir cualquier otra función de activación. La función del perceptrón es simplemente la ponderación de cada señal de entrada con su respectivo peso, pasando el resultado a la función de activación.

$$J(\vec{w}) = \sum_{i=1}^m (h_{\vec{w}}(\mathbf{e}^{(i)}) - l^{(i)})^2$$

Ilustración 16: Función de error teniendo en cuenta la función de activación (Guerra, El perceptrón, s.f.)

En segundo lugar, calculamos la función de error para cada muestra como se aprecia en la Ilustración 16 (Guerra, El perceptrón, s.f.), usando el resultado de la función del perceptrón calculada en el paso anterior que tiene en cuenta la función de activación, al contrario de la función de error de ejemplo que se mostraba en la Ilustración 14.

$$\vec{w}_{t+1} := \vec{w}_t - \gamma \frac{\partial J(\vec{w})}{\partial \vec{w}}$$

Ilustración 17: Actualización de pesos tras cada iteración del entrenamiento (Guerra, El perceptrón, s.f.)

El aprendizaje se hace en este último paso. Al hacer la derivada parcial del vector de pérdidas (el vector con los resultados de las funciones de error) con el vector de los pesos obtenemos la “dirección” a la que tienen que dirigirse los pesos para descender por el gradiente y minimizar los fallos. La velocidad del aprendizaje viene dada por “ γ ”, denominada tasa de aprendizaje. A mayor tasa de aprendizaje, más bruscos serán los cambios en los pesos en cada iteración, y mayor es el riesgo de no solo no descender por el gradiente, sino retroceder y perder precisión con cada iteración.

5.3.2. La red neuronal

La combinación de las neuronas es la solución a los problemas de clasificación que no son separables linealmente, y nos abre las puertas a un amplio abanico de posibilidades.

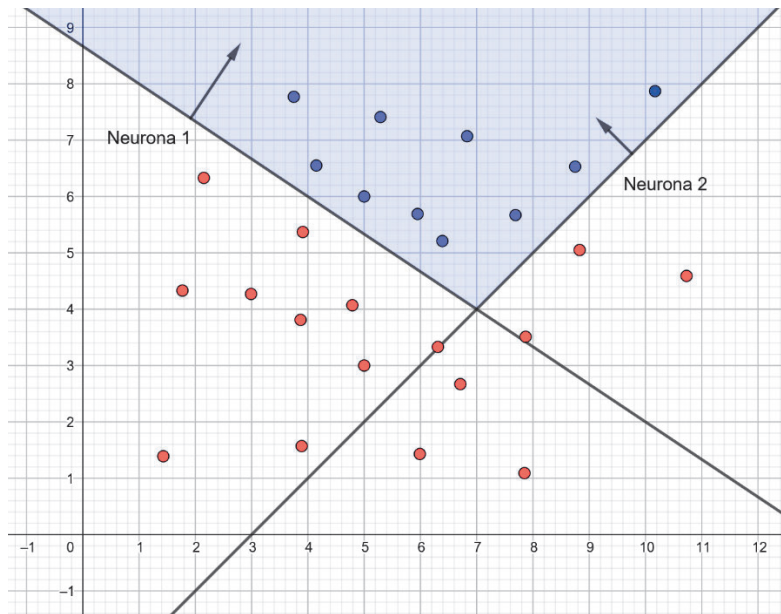


Ilustración 18: Ejemplo de una red neuronal solucionando un problema de clasificación no separable linealmente (Guerra, Redes neuronales 1, s.f.)

En la Ilustración 18 (Guerra, Redes neuronales 1, s.f.) podemos apreciar un ejemplo de un problema de clasificación no separable linealmente siendo solucionado por la unión de dos neuronas, más otra neurona que nos permita especificar la región concreta que queremos considerar válida.

Cuanto más neuronas contenga una red, más compleja puede llegar a ser la solución para el problema de clasificación. Sin embargo, debe prevalecer la precaución, ya que si tenemos neuronas de más se puede dar “overfitting”, que significa que nuestra red se ha vuelto tan buena en clasificar las muestras que le dimos en el entrenamiento, que al darle muestras que no ha visto nunca fallará con mucha más frecuencia por las pequeñas diferencias que pueden llegar a tener estas.

5.3.2.1. El aprendizaje de las redes neuronales

El proceso de aprendizaje de las redes neuronales es similar al de una sola neurona, pero como los resultados de cada capa de neuronas están vinculados a los resultados de cada capa anterior, es necesario un sistema que las vincule y nos permita saber qué neuronas son las que mayor variación en los pesos requerirán para el descenso por el gradiente.

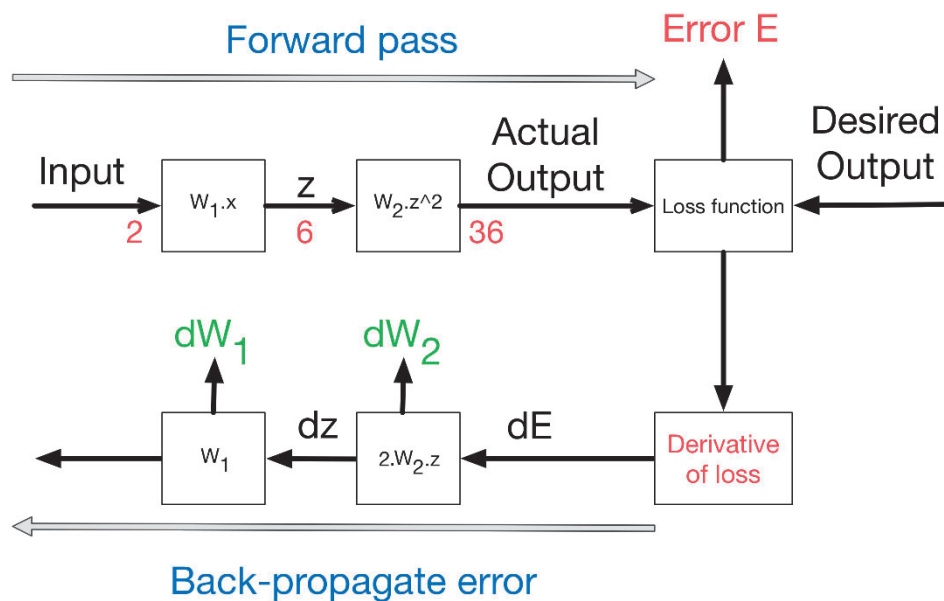


Ilustración 19: Propagación hacia adelante y hacia atrás en una red neuronal (Moawad, 2018)

El sistema que nos permite realizar dicha tarea es la propagación hacia atrás, o “backpropagation”, que se muestra de forma abstracta en la Ilustración 19 (Moawad, 2018). Sin entrar demasiado en detalles, el proceso es el siguiente:

1. Se introducen los datos de entrada.
2. Obtenemos el resultado de la clasificación, propagando hacia adelante los resultados de cada capa.
3. Obtenemos el resultado de la función de error.
4. Derivamos la función de error.
5. Propagamos hacia atrás los errores, actualizando los pesos apropiadamente en cada capa.

Los cálculos realizados en la propagación hacia atrás resultan complejos y, aunque conviene conocerlos para saber de mejor manera qué ocurre cuando entrenamos nuestras redes neuronales, no es estrictamente necesario dominarlos, ya que prácticamente todas las librerías de alto nivel en el mercado que nos permiten manejar esta tecnología entrenan las redes por nosotros.

5.3.3. La red neuronal convolucional

Las redes neuronales convolucionales son una de las mayores innovaciones en el sector de la visión por computador desde sus orígenes. Se trata de una red neuronal multicapa clásica con ciertos añadidos que permiten que tenga resultados excelentes al clasificar imágenes, y está basada en cómo funcionan las neuronas en el córtex visual de los animales (Chatterjee, 2019).

Así pues, de la misma forma que nosotros vemos la imagen de un perro y somos capaces de distinguir que lo es en vez de pensar que es otro animal gracias a sus rasgos distintivos que tenemos grabados en nuestra mente, las redes neuronales convolucionales también descomponen las imágenes en rasgos, que a través de la comparación con los demás rasgos de las imágenes de perros que han visto hasta ahora les permite clasificarlas correctamente.

Tradicionalmente, y como veremos más adelante, los rasgos en los que se suelen fijar las redes neuronales convolucionales de cara a clasificar imágenes suelen ser factores de bajo nivel como bordes y curvas en las capas más tempranas, que van combinándose y convirtiéndose en rasgos más complejos como pueden ser las patas de los perros o sus caras, en este ejemplo concreto de las imágenes de perros.

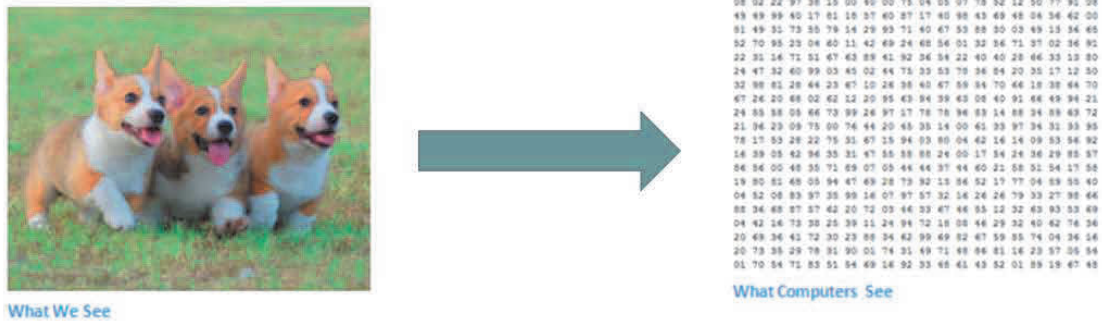


Ilustración 20: Como vemos nosotros las imágenes frente a como las ven los ordenadores (Chatterjee, 2019)

Como podemos observar en la Ilustración 20, los ordenadores ven las imágenes de forma muy distinta a como las vemos nosotros. Para estos, las imágenes son vectores tridimensionales de números con valores comprendidos entre 0 y 255. Esta tridimensionalidad y estos rangos de números se deben al sistema RGB, por el cual para cada píxel tenemos tres números que describen la intensidad de cada color (rojo, verde o azul) que habita en este. Así pues, una imagen en escala de grises resulta ser un vector bidimensional, ya que no es necesario más que un número para cada píxel para describir el color.

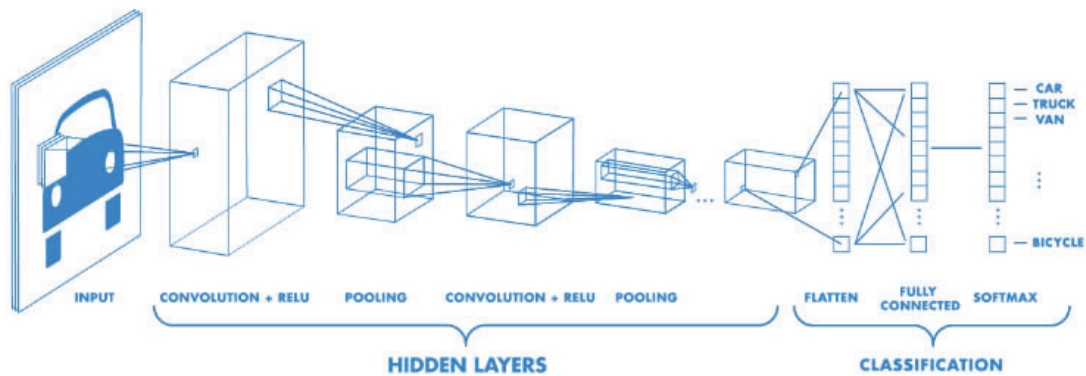


Ilustración 21: Estructura típica de una red neuronal convolucional (Chatterjee, 2019)

Por lo general, las redes neuronales convolucionales suelen tener una arquitectura estandarizada, que se compone de una capa de entrada por la que se introduce la imagen, varias capas convolucionales que extraen rasgos cada vez más complejos de las imágenes (que para ello pasan por un proceso de “pooling” que explicaremos a continuación), y por último una red neuronal “fully connected”, caracterizada por el hecho de que cada neurona de cada capa está conectada con cada neurona de la capa siguiente, y que se encarga de analizar estos rasgos de alto nivel y dar una clasificación final.

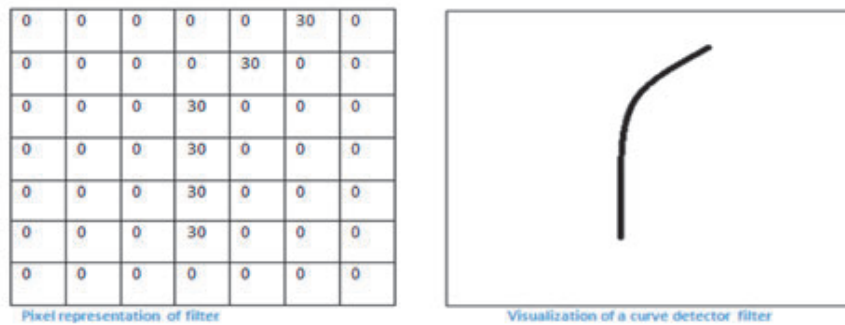


Ilustración 22: Visualización de un filtro de 7x7 (Chatterjee, 2019)

La forma que tienen las redes neuronales convolucionales de detectar rasgos en las imágenes es mediante filtros, como el que se muestra en la Ilustración 22. Los filtros están especializados en un tipo de forma, y tienen la peculiaridad de que también pueden ser entrenados, así que no tenemos que preocuparnos de ellos, tomarán los valores adecuados de forma automática.

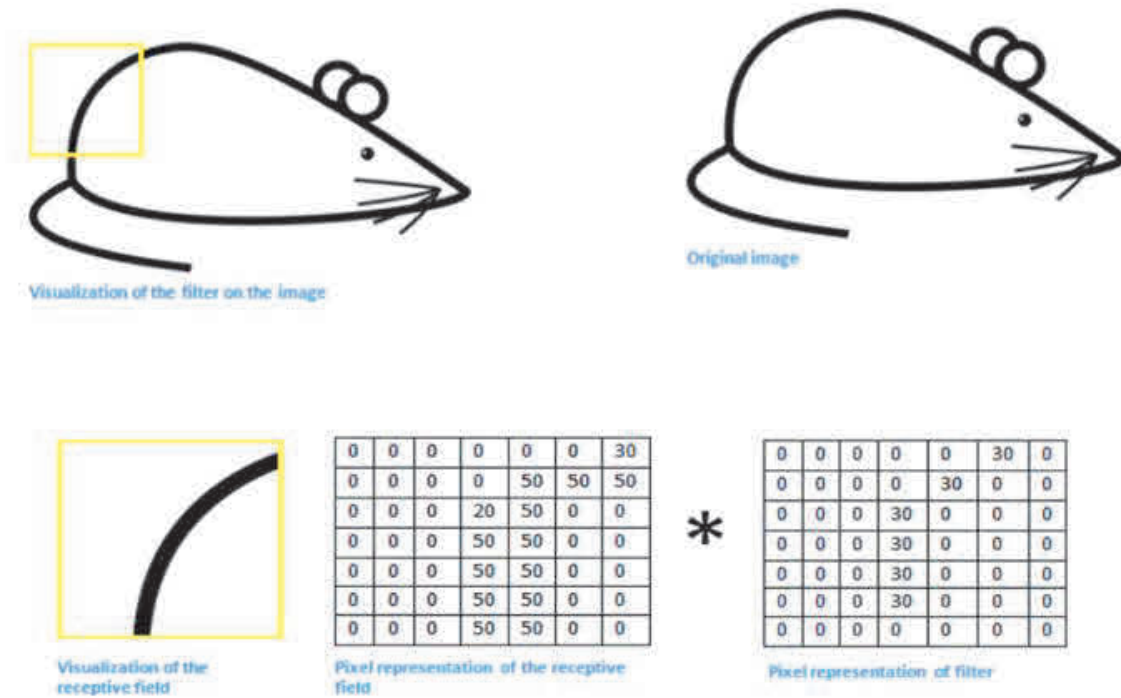


Ilustración 23: Ejemplo del funcionamiento de un filtro (Chatterjee, 2019)

Para el correcto funcionamiento de este tipo de redes y la apropiada detección de rasgos vamos pasando los filtros por cada subsección posible de la imagen, realizando multiplicaciones entre el valor de cada píxel y cada celda del filtro, sumando todos los resultados. Cuanto mayor sea el resultado, mayor es la representación de la figura descrita por el filtro en la subsección de la imagen.

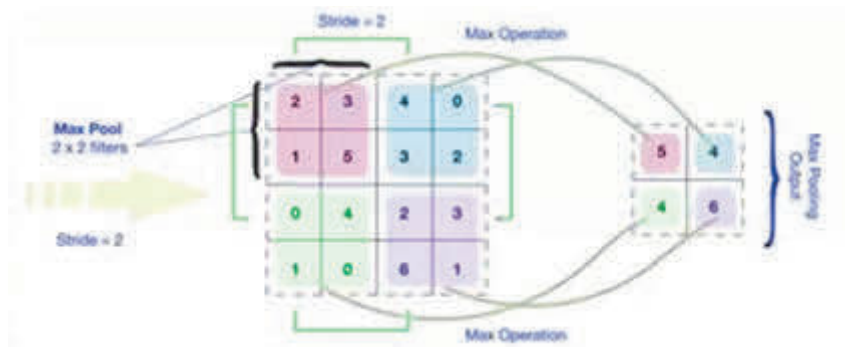


Ilustración 24: Operación de "pooling" (Chatterjee, 2019)

Tras cada capa convolutiva, se realiza el proceso de "pooling", mostrado en la Ilustración 24. Este proceso permite eliminar información redundante, quedándonos sólo con los datos más relevantes, pudiendo así las capas posteriores aplicar sus filtros y encontrar rasgos más complejos. No existe una fórmula perfecta para determinar cuántas parejas de capas convolucionales y pooling son las óptimas, ya que depende de cada caso. Sin embargo, hay numerosas arquitecturas de redes neuronales convolucionales que son muy versátiles y comúnmente usadas, como la arquitectura VGG-16. De esta forma, no siempre es necesario reinventar la rueda trasteando con la arquitectura de la red, a veces una prediseñada realiza apropiadamente la tarea que necesitamos (Thakur, 2019).

Como las capas convolucionales trabajan con vectores multidimensionales, es necesario convertirlos a unidimensionales antes de pasárselos a las capas "fully connected". Para ello, usamos la capa "flatten", que realiza esta conversión. Tras esto, los rasgos de alto nivel generados por las capas convolucionales son analizados por las capas finales y se devuelve como resultado la clasificación final.

6. Capítulo 6: Licencias del Software utilizado

6.1. Licencia MIT

La licencia MIT (The MIT License, s.f.) es una licencia de software permisiva usada en numerosos proyectos de código abierto, que otorga los derechos de uso, copia, modificación, unión, distribución, sublicencia y venta del software. La única condición que plantea es aplicar la misma licencia en todas las copias del software, ya sean parciales o completas.

Keras, la librería de Deep Learning para *Python*, usa esta licencia.

6.2. Licencia Apache 2.0

La licencia Apache 2.0 (Apache License, s.f.) es otra licencia de software permisiva similar a la licencia MIT, pero con ciertas peculiaridades. Todas las modificaciones que se realicen sobre código que esté bajo esta licencia se deben notificar. Además, la licencia Apache también

impone ciertas restricciones sobre el uso de los nombres: un subproyecto que herede código bajo la licencia Apache no puede usar el nombre del proyecto original excepto para explicar su origen.

Tensorflow, la plataforma en la que se basa *Keras*, usa esta licencia.

6.3. Licencia GPL

La licencia GPL (GNU General Public License, s.f.) es también una licencia de software permisiva similar a las anteriores, pero con un énfasis en la obligación de mantener abierto el código de los posibles subproyectos que deriven de un proyecto bajo esta licencia. Se basa, en cierto modo, en la forma de pensar de “si yo he publicado el código fuente de mi proyecto, tú también”.

Git, el software de control de versiones que usamos, se encuentra bajo esta licencia.

7. Capítulo 7: Análisis

7.1. Actores

7.1.1. Programador informático

El programador informático es el único actor que se contempla en la aplicación en su estado actual. Concretamente, se refiere a los programadores que forman parte del equipo de digitalización de cursos de la empresa. Dicho actor tendrá acceso a todos los casos de uso de la aplicación, que se detallarán a continuación.

Especificamos su rol de programador para reiterar que la aplicación no busca reemplazar a los programadores informáticos que se encargan de digitalizar cursos: para realizar el proceso de digitalización de cursos formativos se sigue requiriendo intervención humana y conocimientos técnicos informáticos.

7.2. Casos de uso

7.2.1. Diagrama de casos de uso

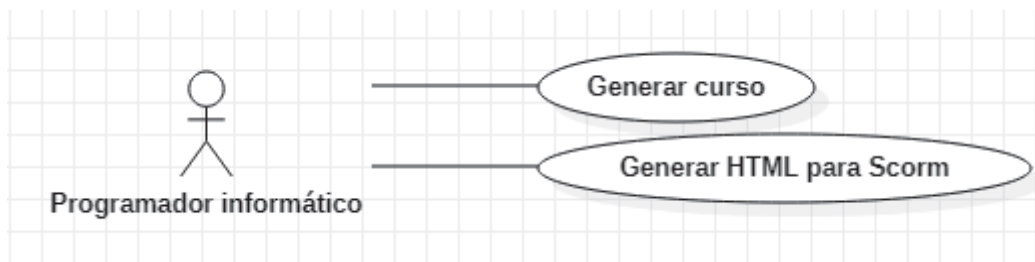


Ilustración 25: Casos de uso de la aplicación

Como podemos observar en la Ilustración 25, los casos de uso posibles en la aplicación son dos: generar los ficheros de un curso o generar los ficheros HTML de un curso de cara a la conversión de este a formato SCORM.

De la misma forma que se podrían hacer más casos de uso para cada uno de los subproblemas que se solucionan a través de esta aplicación, hemos optado por simplificar el proceso y congregarlo todo en el caso de uso de “Generar curso”, para que los usuarios no tengan que perder tiempo realizando más interacciones con la aplicación y hacer más ameno y sencillo el proceso de aprendizaje.

El caso de uso de “Generar HTML para Scorm” se mantiene de forma independiente, ya que no siempre los cursos se convierten a Scorm (los cursos alojados en smartly.es, por ejemplo, se mantienen en su formato original).

7.2.2. Especificación de casos de uso

A continuación, mostramos las especificaciones detalladas de ambos casos de uso.

CASO DE USO		Generar curso
Descripción	El usuario genera el código del curso a partir de las imágenes de las diapositivas del manual correspondiente.	
Actores	Programador informático	
Precondiciones	<ul style="list-style-type: none"> El usuario dispone de las imágenes de las diapositivas del manual en formato JPG y nombradas adecuadamente (“DiapositivaX.JPG”, donde X es el número de la diapositiva) El usuario tiene el repositorio de plantillas almacenado localmente en su dispositivo 	
Flujo normal	Paso	Acción
	1	El usuario selecciona las siguientes rutas: <ul style="list-style-type: none"> Repositorio de plantillas Carpeta con las imágenes de las diapositivas del manual Manual en formato PPT Carpeta donde desea que se genere el curso.
	2	El usuario introduce los números de las diapositivas donde empieza cada módulo, además del número total de módulos
	3	El usuario selecciona entre las siguientes opciones: <ul style="list-style-type: none"> No extraer las imágenes del manual

		<ul style="list-style-type: none"> • Extraer las imágenes del manual en escala de grises • Extraer las imágenes del manual en color
	4	El usuario selecciona entre las siguientes opciones: <ul style="list-style-type: none"> • No extraer los textos de las diapositivas • Extraer los textos e insertarlos en los ficheros de código generados • Extraer los textos e insertarlos en ficheros aparte
	5	El usuario selecciona si el curso usa el modelo antiguo de plantillas o el nuevo.
	6	El usuario selecciona el botón “Generar curso”
Postcondiciones	El código del curso se ha generado correctamente y se ha almacenado en la ruta especificada.	
Variaciones	Paso	Acción
Extensiones		
Excepciones	1	El usuario introduce alguna ruta incorrecta

Tabla 2: Especificación del caso de uso "Generar curso"

Como podemos observar en la Tabla 2, el caso de uso “Generar curso” requiere una especificación densa, debido a las numerosas opciones que puede configurar el usuario.

En primer lugar, para poder generar un curso es necesario que ya se haya realizado la exportación del manual a imágenes, que si se usa el método facilitado por PowerPoint se realizará automáticamente en el formato apropiado. Por otra parte, el usuario debe tener localmente el repositorio de plantillas, aunque realmente sólo es necesario el fichero model/slider.php, que es el que accedemos para extraer el código de las plantillas. De cualquier forma, este tampoco es un impedimento, ya que los programadores del equipo de digitalización siempre cumplirán este requisito y siempre tendrán el fichero actualizado.

Si el usuario cumple las precondiciones, debe introducir las rutas de las carpetas y ficheros pedidos, además de especificar el número de módulos y los números de las diapositivas donde empieza cada uno. Esta quizás es la tarea más monótona y tediosa, pero no es nada comparado con tener que digitalizar el curso manualmente. De todas formas, en el futuro se podría implementar un sistema de detección automática de diapositivas de inicio para eliminar este aspecto negativo de la experiencia de uso.

Tras esto, el usuario debe decidir si el usuario quiere extraer las imágenes y/o los textos del manual, y en caso de que sí, escoger si los desea en escala de grises o en color en caso de las imágenes, y si los desea en los propios ficheros de código generados o en ficheros aparte, en el caso de los textos. La funcionalidad de elegir donde se insertan los textos fue implementada a raíz de una sugerencia de mejora perteneciente de un compañero del equipo de digitalización de cursos.

Por último, el usuario selecciona si el curso usa el modelo antiguo de plantillas o el nuevo, y presiona el botón de “Generar curso”. Tras esto, el curso se habrá generado en la ruta especificada.

CASO DE USO		Generar HTML para Scorm
Descripción	El usuario descarga los ficheros HTML necesarios para la conversión de un curso a Scorm	
Actores	Programador informático	
Precondiciones	<ul style="list-style-type: none"> El usuario dispone de conexión a internet 	
Flujo normal	Paso	Acción
	1	El usuario selecciona la ruta donde desea que se le generen los archivos.
	2	El usuario configura la URL en la que se encuentra el curso.
	3	El usuario introduce el número de módulos del curso.
	4	El usuario selecciona si los nombres de los ficheros usen la nomenclatura de la FULP (versión FULP) o no.
	5	El usuario selecciona si el curso usa el modelo antiguo de plantillas o el nuevo.
	6	El usuario selecciona el botón "Crear archivos"
Postcondiciones	Los ficheros HTML se han descargado en la ruta seleccionada con la nomenclatura seleccionada	
Variaciones	Paso	Acción
Extensiones		
Excepciones	1	El usuario introduce alguna ruta incorrecta
	4	No existe un curso asociado a la URL introducida

Tabla 3: Especificación del caso de uso "Generar HTML para Scorm"

El caso de uso cuya especificación podemos encontrar en la Tabla 3 es más reducido, ya que dispone de muchas menos opciones de personalización que el caso de uso anterior. Los usuarios simplemente escogen la ruta donde desean los archivos, especifican la URL del curso, introducen el número de módulos que contiene el mismo y seleccionan la nomenclatura de los ficheros deseada y ya pueden generar los ficheros deseados. De cara a la URL del curso, el sistema autocompleta una ruta por defecto en la que se suelen alojar los cursos, pero es modificable por el usuario en caso de que fuera incorrecta en algún caso particular.

Por último, al contrario que el caso de uso de "Generar curso", aquí sí es necesaria la conexión a internet, pero este requisito no resulta ser ningún problema para los integrantes del equipo de digitalización.

8. Capítulo 8: Diseño

8.1. Diseño de la arquitectura de la red neuronal convolucional

Con este proyecto nos enfrentamos a un reto de clasificación de imágenes, en el que hay un gran número de clases (hay más de 100 plantillas, pero actualmente sólo se usan 56). Además,

actualmente se están desarrollando nuevas plantillas para un nuevo modelo de cursos (lo llamaremos **“Modelo GDT”** de aquí en adelante, en referencia a los nuevos cursos de Gestión del Tiempo), que actualmente sólo tiene 27 plantillas que se usan activamente, pero que próximamente irá aumentando. Así pues, para clasificar las imágenes haremos uso de una red neuronal convolucional construida y entrenada por nosotros usando la librería Keras, que nos facilita notablemente estos procesos.

Nos enfrentamos a dos conjuntos de datos compuestos por plantillas con aspectos visuales distintos, pero la arquitectura de la CNN que usaremos será la misma. Posteriormente comprobaremos si esta decisión fue la correcta o si es necesario que definamos otra CNN con arquitectura distinta para que ambas redes clasifiquen apropiadamente las imágenes de cada modelo de plantillas.

Por consiguiente, hemos optado por la siguiente arquitectura para nuestra CNN:

- **Capa convolucional 2D**
 - 32 filtros
 - Tamaño del kernel 3x3
 - Función de activación ReLU
- **Capa MaxPooling 2D de 2x2**
- **Capa convolucional 2D**
 - 64 filtros
 - Tamaño del kernel 3x3
 - Función de activación ReLU
- **Capa MaxPooling 2D de 2x2**
- **Capa convolucional 2D**
 - 128 filtros
 - Tamaño del kernel 3x3
 - Función de activación ReLU
- **Capa MaxPooling 2D de 2x2**
- **Aplanamiento de los datos (Flatten)**
- **Capa Dropout de 0.2**
- **Capa FullyConnected (Dense)**
 - 512 unidades
 - Función de activación ReLU
 - Regularización del kernel L2 de 0.01
 - Regularización de sesgo (bias) L2 de 0.01
- **Capa Dropout de 0.4**
- **Capa FullyConnected (Dense)**
 - 1024 unidades
 - Función de activación ReLU
 - Regularización del kernel L2 de 0.01
 - Regularización de sesgo (bias) L2 de 0.01
- **Capa FullyConnected (Dense)**
 - 56 unidades (o 27 en el Modelo GDT)
 - Función de activación Softmax

Las capas convolucionales son ideales para extraer las características y rasgos de las imágenes, que luego son procesadas por las capas *FullyConnected* y desembocan en la capa *Softmax*, que devuelve la predicción final. Las capas *dropout* y las técnicas de regularización nos permiten combatir el “*overfitting*”, para que la red clasifique correctamente tanto el conjunto de entrenamiento como el conjunto de validación.

8.2. Diseño de los conjuntos de datos usados para el entrenamiento de la CNN

Para el diseño de los conjuntos de datos hemos extraído las imágenes de las diapositivas correspondientes a los primeros manuales desarrollados para el programa Digitalizados de la Fundación Universitaria de Las Palmas, y los primeros manuales desarrollados para los nuevos cursos de Gestión del Tiempo propios de la empresa (para el conjunto de datos del modelo GDT).

De cara al almacenamiento y la organización de los conjuntos de datos, hemos optado por almacenarlos en Google Drive y organizados en carpetas cuyos nombres representan los nombres de las clases que la red clasificará. En ese caso, dichos nombres serán los números de identificación de las plantillas.

Almacenarlos en Google Drive nos otorga diversas ventajas: podemos acceder a los ficheros fácilmente en Google Colab gracias a la integración entre ambas plataformas, prevenimos posibles fallos de nuestros dispositivos locales que nos pudieran hacer perder nuestros conjuntos de datos, y nos facilita tener los datos de forma centralizada y accesible desde cualquier dispositivo, todo esto de forma gratuita.

Tras extraer las imágenes de los manuales, las clasificamos manualmente consultando el repositorio de plantillas, y las fuimos almacenando en las carpetas correspondientes en Google Drive. Originalmente las insertamos todas en el conjunto de entrenamiento, y posteriormente escogimos un subconjunto de estas y las exportamos de forma que constituyesen el conjunto de validación. En total, nuestros conjuntos de imágenes tienen las siguientes dimensiones:

- **Modelo estándar (o modelo antiguo)**
 - 705 imágenes en el conjunto de entrenamiento
 - 336 imágenes en el conjunto de validación
- **Modelo GDT**
 - 254 imágenes en el conjunto de entrenamiento
 - 82 imágenes en el conjunto de validación

Con respecto al número de imágenes de cada conjunto, pretendimos tener una proporción 70/30 para los conjuntos de entrenamiento y validación, respectivamente. Así pues, el conjunto de validación del modelo estándar representa un 32.27% de las imágenes totales y el del modelo GDT un 24.4%.

El hecho de que el modelo GDT tenga una proporción más baja de imágenes de validación no es casualidad: este modelo es de origen muy reciente (con muy pocos manuales desarrollados

por aquel entonces), y nos era imposible reunir un conjunto de datos de dimensiones notables (solo disponíamos de 330 imágenes). Así pues, nos resultaba de mayor importancia que, dada la escasez total de imágenes, la red tuviese un mayor número de imágenes de entrenamiento. Además, las plantillas del modelo GDT están en desarrollo activo y pueden cambiar con facilidad o incorporarse nuevas al repositorio rápidamente. Por esta razón, cabe esperar que el modelo GDT no tenga tanta precisión como el modelo estándar en las clasificaciones, al menos de forma temporal hasta que se ponga más énfasis en la creación de nuevos cursos en vez del desarrollo de nuevas plantillas.

8.3. Diseño de la interfaz gráfica de usuario

De cara al diseño de la interfaz gráfica de usuario, hemos realizado algunos mockups para establecer la posición de cada componente y facilitarnos el desarrollo. Para comenzar, mostramos el mockup para el caso de uso de “Generar curso”:

El mockup muestra una interfaz de usuario con dos pestañas: "Generar curso" (seleccionada) y "Generar HTML para Scorm".

Dentro de la pestaña "Generar curso", hay un formulario con los siguientes campos:

- Cuatro campos de texto con el prefijo "Ruta del" (repositorio, imágenes, manual, donde generar el curso) y un botón "Seleccionar ruta..." a la derecha.
- Cuatro campos de entrada de número con prefijos "Número total de módulos", "Diapositivas del módulo 1", "Diapositivas del módulo 2" y "Diapositivas del módulo 3". Los valores actuales son 3, 45, 60 y 75, respectivamente.
- Un campo "Extraer imágenes del manual" con un menú desplegable que muestra "Extraer en color".
- Un campo "Extraer textos del manual" con un menú desplegable que muestra "No extraer".
- Un botón "Generar curso" y una casilla de verificación "Usar modelo GDT" que está marcada.

Ilustración 26: Mockup del caso de uso "Generar curso"

Como podemos observar en la Ilustración 26, la sección en la interfaz de usuario donde se especifican las rutas sería la primera que viese el usuario, correspondiéndose al primer paso que debe realizar para llevar a cabo la generación de un curso según la especificación del caso

de uso. Los botones de “Seleccionar ruta” abrirían una ventana del explorador del sistema operativo donde se pudiese escoger una carpeta o un fichero, en el caso del manual.

A continuación, se le presenta al usuario una sección donde puede introducir la información sobre los módulos del curso. Al introducir el número total de módulos, el número de filas donde se puede especificar la diapositiva de comienzo de cada módulo se ajustaría al total de estos automáticamente. De esta forma, si el usuario seleccionase que el número total de módulos fuese veinte, automáticamente se le mostraría el mismo número de campos donde seleccionar las diapositivas de comienzo correspondientes.

Por último, al usuario se le presentan las opciones relacionadas con las extracciones de imágenes y textos del manual, además de un “checkbox” donde se puede elegir el modelo de plantillas usado. Tras esto, el sistema ya está listo para generar el curso según el usuario presione el botón de generación.

Los elementos usados en la interfaz de usuario son comúnmente usados en todo tipo de interfaces y son considerados un estándar, así que a los usuarios finales no les costará interpretar la interfaz. Además, las secciones están situadas de tal forma que el flujo visual se corresponde con el flujo operacional del caso de uso.

Ilustración 27 muestra un mockup de la interfaz de usuario para el caso de uso "Generar HTML para Scorm". La interfaz está organizada en una pestaña superior con dos opciones: "Generar curso" y "Generar HTML para Scorm". El formulario principal contiene los siguientes elementos:

- Un campo de texto etiquetado "Ruta donde generar los archivos" con un botón "Seleccionar ruta..." a su derecha.
- Un campo de texto etiquetado "Nombre del curso" con un botón "Introduce un nombre..." a su derecha.
- Un campo de texto que muestra el URL "https://urlpordefecto.com/nombre_del_curso".
- Un campo de texto etiquetado "Número total de módulos" con un control de espín que muestra el número "3".
- Un botón "Crear archivos" situado a la izquierda de un checkbox etiquetado "Versión FULP", que está marcado.

Ilustración 27: Mockup del caso de uso "Generar HTML para Scorm"

Por otra parte, como podemos observar en la Ilustración 27, el caso de uso de “Generar HTML para Scorm” dispone de una interfaz de usuario más escueta y comprimida, correspondiéndose al menor número de acciones que realizan los usuarios en la misma.

La estructura de la interfaz con respecto al posicionamiento de los elementos es similar al caso de uso anterior, estando situados en primer lugar los campos relacionados con los primeros pasos a realizar en el caso de uso, y en último lugar los que representan los pasos finales.

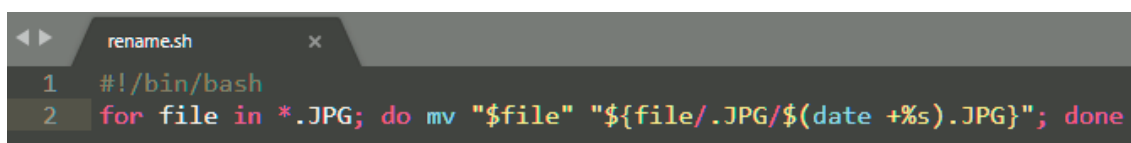
Para finalizar, cabe destacar que cada caso de uso se encuentra en una pestaña propia de la interfaz, cuya visibilidad se puede alternar a través del menú superior. Esta modularización ayuda al usuario a entender más fácilmente la interfaz y centrarse más en la tarea a llevar a cabo, gracias a la menor saturación de información que se le muestra.

9. Capítulo 9: Desarrollo

9.1. Estructura del proyecto

Como hemos mencionado anteriormente, el proyecto consistirá en dos hojas de Google Colab (una para cada modelo) en las que procesaremos los conjuntos de datos y entrenaremos las redes neuronales, y además una aplicación de escritorio realizada en Python 3.7 que será la que usen los usuarios finales.

9.2. Conjuntos de datos de imágenes

A screenshot of a terminal window with a dark background. The title bar shows 'rename.sh' and a close button. The terminal content is as follows:

```
1 #!/bin/bash
2 for file in *.JPG; do mv "$file" "${file}/.JPG/${date +%s}.JPG}"; done
```

Ilustración 28: Script bash para renombrar todos los ficheros de una carpeta

Durante el proceso de recopilación y almacenamiento en Google Drive de las imágenes nos encontramos con el siguiente problema: cuando subíamos las imágenes a Google Drive, si existían imágenes en las mismas carpetas con el mismo nombre, se sustituían por las ya existentes en vez de añadirse con otro nombre. Esto dañaba notablemente la integridad de los conjuntos de datos, así que nos vimos obligados que crear el pequeño script bash que vemos en la Ilustración 28 para renombrar todas las imágenes de cada curso que íbamos clasificando a unos nombres que no existiesen previamente en las carpetas online. Esto lo conseguimos fácilmente anexando la fecha actual a los nombres de cada uno de los ficheros usando un iterador de ficheros JPG que llame al comando “mv” con la función “date”.

Una vez hemos recopilado las imágenes y las hemos almacenado en las carpetas correspondientes en Google Drive, ya podemos comenzar a procesarlas en las hojas de Google Colab para entrenar a las redes neuronales. Para transformar los conjuntos de datos de imágenes a un formato que puedan procesar correctamente las redes neuronales, usamos los generadores de datos de imágenes de *Keras (ImageDataGenerator)*. Concretamente, el código utilizado es el siguiente:

```
datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0,
    zoom_range=0,
    horizontal_flip=False,
    validation_split=0.0)
```

Los parámetros introducidos cumplen las siguientes funciones:

- **Rescale:** Conversión de los valores de las imágenes para que se sitúen entre 0 y 1 en vez de entre 0 y 255.
- **Shear range:** No se realiza ninguna torsión en las imágenes.
- **Zoom range:** No se realiza ningún zoom en las imágenes.
- **Horizontal flip:** No se voltea ninguna imagen.
- **Validation split:** No se reserva ninguna de las imágenes para el conjunto de validación (ya que cada conjunto tiene sus imágenes en carpetas distintas)

Para cada conjunto (*train* y *validation*) usaremos el método *flow_from_directory* para ir leyendo las imágenes, de la siguiente forma:

```
batch_size = 16
imgsize = 150
# Carpeta con las imágenes de entrenamiento
train_data_dir = '/content/drive/My Drive/Colab Notebooks/datasets/diapos/train'
# Carpeta con las imágenes de validación
validation_data_dir = '/content/drive/My Drive/Colab Notebooks/datasets/diapos/validation'
train_generator = datagen.flow_from_directory(
    train_data_dir,
    target_size=(imgsize, imgsize),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale',
    shuffle=True)

validation_generator = datagen.flow_from_directory(
    validation_data_dir,
    target_size=(imgsize, imgsize),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale',
    shuffle=True)
```

Los parámetros introducidos cumplen las siguientes funciones:

- **Target size:** Redimensionamiento de las imágenes a 150x150px cada una.
- **Batch size:** El tamaño de los lotes de entrenamiento será de 16 imágenes.
- **Class mode:** Indicamos que las etiquetas están codificadas usando *One hot*.
- **Color mode:** Convertimos las imágenes a escala de grises. (En el modelo GDT, las imágenes se mantienen en color)
- **Shuffle:** Aleatorizamos el orden en el que se procesan las imágenes.

9.3. Entrenamiento de la red neuronal convolucional

Antes de entrenar la red, creamos su arquitectura usando los métodos provistos por *Keras*:

```

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                activation='relu',
                input_shape=(imgsize, imgsize, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_regularizer=l2(0.01),
                bias_regularizer=l2(0.01)))
model.add(Dropout(0.4))
model.add(Dense(1024, activation='relu', kernel_regularizer=l2(0.01),
                bias_regularizer=l2(0.01)))
model.add(Dense(56, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])

```

La arquitectura se corresponde con la descrita en el apartado 8.1. Esta arquitectura es la misma que la que usamos en el modelo GDT, con la excepción de que la capa *softmax* tiene 27 unidades en vez de 56. Para la función de pérdida, hemos escogido la función de entropía cruzada, y para el optimizador hemos escogido Adam. A continuación, mostramos el resumen de la arquitectura que nos proporciona *Keras*:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 17, 17, 128)	0
flatten_1 (Flatten)	(None, 36992)	0
dropout_1 (Dropout)	(None, 36992)	0

dense_1 (Dense)	(None, 512)	18940416
dropout_2 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1024)	525312
dense_3 (Dense)	(None, 56)	57400
=====		
Total params: 19,615,800		
Trainable params: 19,615,800		
Non-trainable params: 0		

El siguiente paso ya es entrenar las redes. Lo realizamos de la siguiente forma:

```

epochs = 12
step_train = (train_generator.n//epochs)
step_validation = (validation_generator.n//epochs)
modelHistory = model.fit_generator(
    train_generator,
    steps_per_epoch=step_train,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=step_validation
)

```

Para el modelo antiguo realizamos 12 épocas de entrenamiento, y para el nuevo 25. Para determinar el número de lotes de imágenes que representan una época, dividimos el número total de imágenes en cada conjunto por el número de épocas establecido.

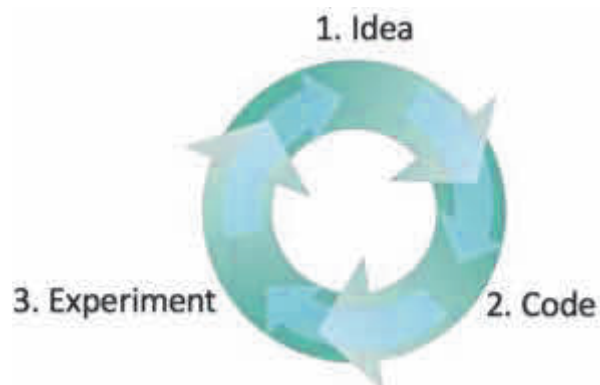


Ilustración 29: Proceso iterativo del Machine Learning (Dan Clark, 2018)

Como podemos observar en la Ilustración 29 (Dan Clark, 2018), el proceso de desarrollo cuando se trabaja en *Machine Learning* es puramente iterativo y basado en la prueba y el error. La mejor manera de conseguir una red que se ajuste a nuestras necesidades es mediante la sucesión de experimentaciones con diferentes arquitecturas e hiperparámetros. De esta forma, en caso de que una prueba falle (es decir, que no nos sirva para nuestros propósitos), formulamos cambios sobre su arquitectura, los materializamos a través de nuevo código y en un nuevo entrenamiento de la red y probamos de nuevo si satisface nuestras necesidades.

Así pues, la arquitectura por la que hemos optado es meramente una prueba: si cumple nuestras expectativas la mantenemos, y si no realizamos una nueva iteración.

A continuación, adjuntamos los resultados de entrenamiento de la red basada en el modelo antiguo:

```
Epoch 1/12
58/58 [=====] - 17s 301ms/step - loss: 7.6390
- accuracy: 0.2289 - val_loss: 3.4487 - val_accuracy: 0.5469
Epoch 2/12
58/58 [=====] - 15s 258ms/step - loss: 3.4368
- accuracy: 0.5791 - val_loss: 2.7463 - val_accuracy: 0.6250
Epoch 3/12
58/58 [=====] - 15s 259ms/step - loss: 2.9680
- accuracy: 0.7174 - val_loss: 2.7784 - val_accuracy: 0.8438
Epoch 4/12
58/58 [=====] - 16s 283ms/step - loss: 2.3811
- accuracy: 0.8368 - val_loss: 1.8198 - val_accuracy: 0.8504
Epoch 5/12
58/58 [=====] - 15s 261ms/step - loss: 1.9977
- accuracy: 0.8806 - val_loss: 1.7846 - val_accuracy: 0.9420
Epoch 6/12
58/58 [=====] - 16s 271ms/step - loss: 1.7811
- accuracy: 0.9047 - val_loss: 1.4907 - val_accuracy: 0.9174
Epoch 7/12
58/58 [=====] - 15s 265ms/step - loss: 1.7960
- accuracy: 0.9009 - val_loss: 1.8306 - val_accuracy: 0.9286
Epoch 8/12
58/58 [=====] - 15s 264ms/step - loss: 1.6590
- accuracy: 0.9233 - val_loss: 1.8153 - val_accuracy: 0.8996
Epoch 9/12
58/58 [=====] - 15s 261ms/step - loss: 1.9420
- accuracy: 0.8964 - val_loss: 1.6062 - val_accuracy: 0.9308
Epoch 10/12
58/58 [=====] - 15s 252ms/step - loss: 1.9502
- accuracy: 0.8949 - val_loss: 2.0436 - val_accuracy: 0.8884
Epoch 11/12
58/58 [=====] - 16s 284ms/step - loss: 1.9229
- accuracy: 0.9146 - val_loss: 1.5538 - val_accuracy: 0.9174
Epoch 12/12
58/58 [=====] - 15s 266ms/step - loss: 1.5258
- accuracy: 0.9529 - val_loss: 1.1972 - val_accuracy: 0.9397
```

Como podemos observar, hemos alcanzado un 95.29% de precisión en el conjunto de entrenamiento y aproximadamente un 94% en el conjunto de validación, resultados muy favorables en ambos casos. En las pruebas reales (detalladas en apartados posteriores) hemos obtenido un 85% de precisión, que, pese a que sigue siendo un resultado excelente, se aleja de la precisión de entrenamiento. Mostramos ahora los resultados del entrenamiento de la red basada en el modelo GDT:

```
Epoch 1/20
```

```
12/12 [=====] - 3s 254ms/step - loss: 14.3042
- accuracy: 0.1198 - val_loss: 10.1674 - val_accuracy: 0.1094
Epoch 2/20
12/12 [=====] - 3s 215ms/step - loss: 8.7397
- accuracy: 0.2447 - val_loss: 7.4548 - val_accuracy: 0.0800
Epoch 3/20
12/12 [=====] - 2s 176ms/step - loss: 6.4191
- accuracy: 0.2656 - val_loss: 6.1291 - val_accuracy: 0.2400
Epoch 4/20
12/12 [=====] - 3s 217ms/step - loss: 5.0230
- accuracy: 0.3368 - val_loss: 4.8994 - val_accuracy: 0.3594
Epoch 5/20
12/12 [=====] - 3s 218ms/step - loss: 4.0655
- accuracy: 0.5526 - val_loss: 3.7264 - val_accuracy: 0.5200
Epoch 6/20
12/12 [=====] - 2s 175ms/step - loss: 3.4966
- accuracy: 0.6354 - val_loss: 4.1804 - val_accuracy: 0.7200
Epoch 7/20
12/12 [=====] - 3s 218ms/step - loss: 2.8968
- accuracy: 0.7895 - val_loss: 2.5642 - val_accuracy: 0.7500
Epoch 8/20
12/12 [=====] - 3s 213ms/step - loss: 2.6630
- accuracy: 0.8211 - val_loss: 2.7510 - val_accuracy: 0.7000
Epoch 9/20
12/12 [=====] - 2s 179ms/step - loss: 2.5261
- accuracy: 0.8368 - val_loss: 2.0302 - val_accuracy: 0.8600
Epoch 10/20
12/12 [=====] - 3s 217ms/step - loss: 2.1980
- accuracy: 0.8947 - val_loss: 2.1106 - val_accuracy: 0.9219
Epoch 11/20
12/12 [=====] - 3s 223ms/step - loss: 2.1331
- accuracy: 0.8802 - val_loss: 1.9652 - val_accuracy: 0.9200
Epoch 12/20
12/12 [=====] - 2s 169ms/step - loss: 2.0258
- accuracy: 0.8789 - val_loss: 1.9481 - val_accuracy: 0.9000
Epoch 13/20
12/12 [=====] - 3s 219ms/step - loss: 1.7931
- accuracy: 0.9263 - val_loss: 2.1354 - val_accuracy: 0.8594
Epoch 14/20
12/12 [=====] - 3s 219ms/step - loss: 1.8534
- accuracy: 0.9219 - val_loss: 1.7687 - val_accuracy: 0.9400
Epoch 15/20
12/12 [=====] - 2s 177ms/step - loss: 1.7536
- accuracy: 0.8947 - val_loss: 1.3333 - val_accuracy: 0.8400
Epoch 16/20
12/12 [=====] - 3s 216ms/step - loss: 1.5276
- accuracy: 0.9368 - val_loss: 1.3162 - val_accuracy: 0.9531
Epoch 17/20
12/12 [=====] - 3s 217ms/step - loss: 1.4562
- accuracy: 0.9105 - val_loss: 1.7108 - val_accuracy: 0.9200
Epoch 18/20
12/12 [=====] - 2s 177ms/step - loss: 1.3893
- accuracy: 0.9526 - val_loss: 1.1701 - val_accuracy: 0.9200
```

```
Epoch 19/20
12/12 [=====] - 3s 218ms/step - loss: 1.3096
- accuracy: 0.9271 - val_loss: 1.1044 - val_accuracy: 0.8906
Epoch 20/20
12/12 [=====] - 3s 215ms/step - loss: 1.1387
- accuracy: 0.9737 - val_loss: 1.0939 - val_accuracy: 0.9400
```

En este caso también hemos obtenido resultados muy positivos, con un 97.37% de precisión en el conjunto de entrenamiento y un 94% en el conjunto de validación, como en el modelo antiguo. En las pruebas reales (detalladas en apartados posteriores) hemos obtenido un sorprendente 96,2% de precisión, un valor superior incluso al obtenido en el conjunto de validación. Sin embargo, hay que tratar con más cautela estos resultados, ya que el entrenamiento se ha realizado con unos conjuntos de imágenes de mucho menor tamaño que los conjuntos del modelo antiguo. Los siguientes gráficos nos ayudarán a visualizar mejor los resultados:

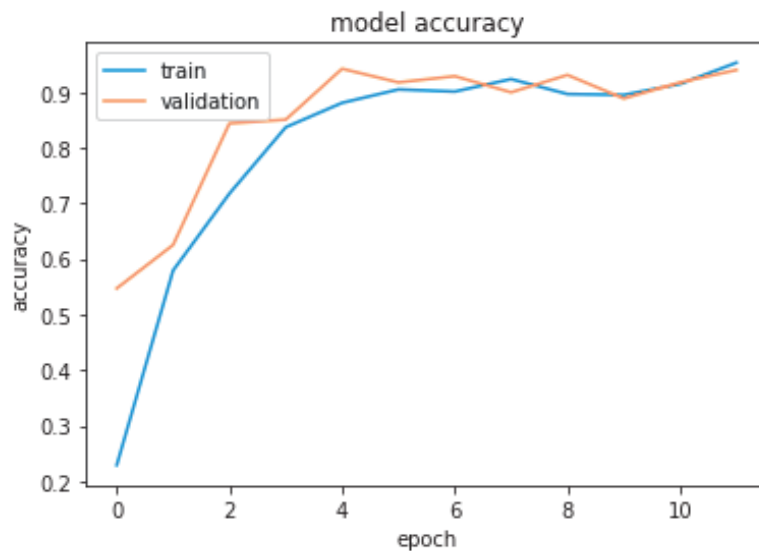


Ilustración 30: Precisión de entrenamiento del modelo antiguo

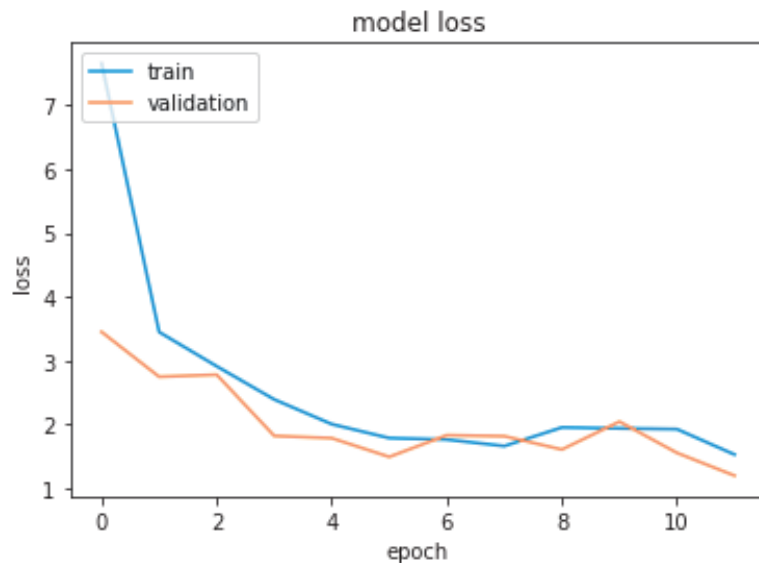


Ilustración 31: Evolución del "loss" del modelo antiguo

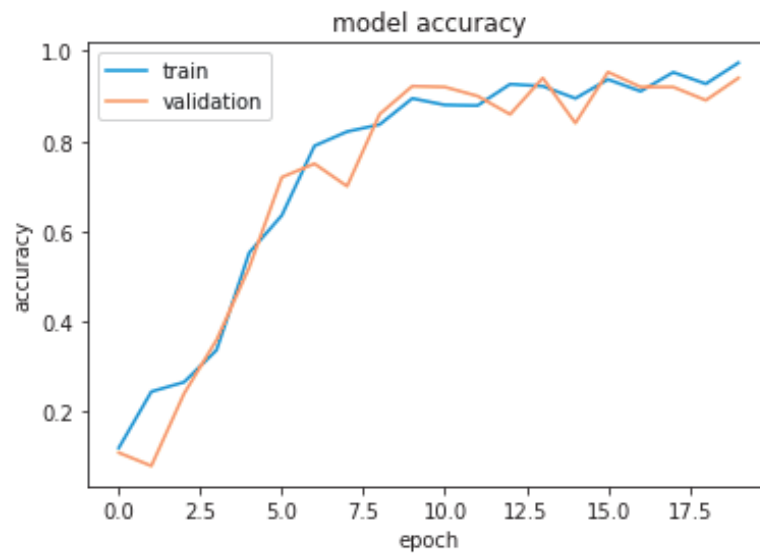


Ilustración 32: Precisión de entrenamiento del modelo GDT

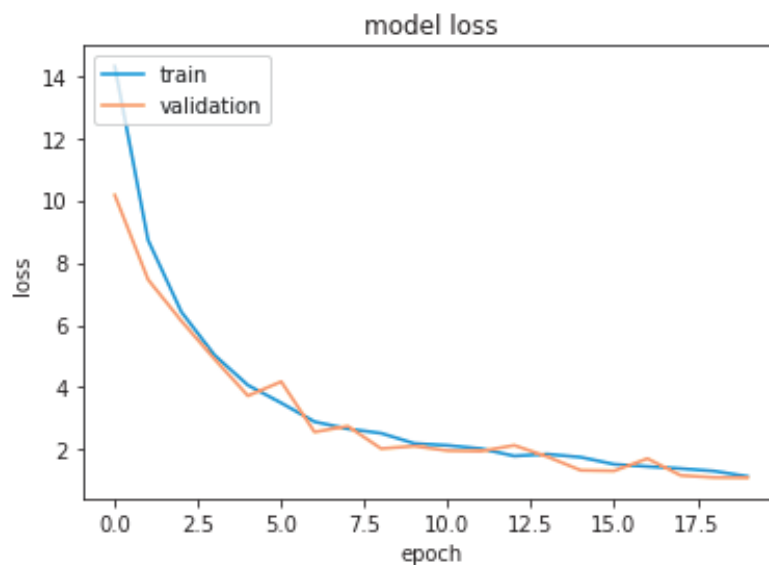


Ilustración 33: Evolución del "loss" del modelo GDT

Como podemos observar, ambos modelos han ido mejorando en precisión progresivamente a medida que se realizaban las épocas. Concretamente, como se representa en la Ilustración 30 y la Ilustración 31, el modelo antiguo comenzó a mostrar resultados aceptables a partir de la tercera época y escasas mejoras a partir de la quinta, a excepción de la última, en la que se observó una mejora de precisión notable. Por otra parte, como se representa en la Ilustración 32 y la Ilustración 33, el modelo GDT comenzó a mostrar resultados aceptables a partir de la octava época, y siguió mejorando su precisión en las épocas sucesivas, aunque a un ritmo menor.

Finalmente, mediante estos gráficos podemos observar que el nivel de *overfitting* obtenido en ambos modelos es bajo, y que los resultados obtenidos son prometedores. Posteriormente, en

la fase de pruebas compararemos la precisión real en los cursos con las precisiones obtenidas en este paso.

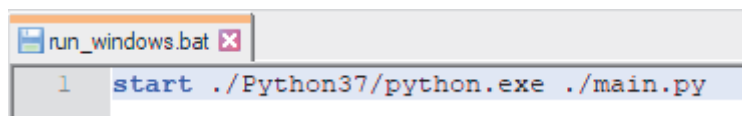
9.4. Desarrollo de la aplicación final

9.4.1. Estructura de la aplicación final

Como hemos mencionado anteriormente, la aplicación final está desarrollada en Python 3.7 y se compone por 7 ficheros que contienen clases que interactúan entre sí, y cuyas funciones explicaremos a continuación:

- **main.py**: Encargada de la inicialización de la aplicación y del funcionamiento de la interfaz gráfica.
- **ai.py**: Lectura de las imágenes y predicciones sobre las mismas usando los modelos que hemos entrenado.
- **generator.py**: Generación de los ficheros de código que corresponden al curso.
- **module.py**: Clase que representa los módulos de los cursos y simplifica el tratamiento de estos.
- **ppt.py**: Extracción de los textos y las imágenes a partir del manual del curso en formato PPT.
- **scorm.py**: Generación de los ficheros HTML para la creación de SCORM.
- **utils.py**: Clase de la que heredan AI y Generator que dispone de diversas utilidades para la generación de directorios y los registros de errores.

El proyecto está pensado para que su instalación y su uso sean lo más sencillas posible. Sin embargo, las herramientas disponibles para compactar el proyecto en un solo archivo ejecutable como *cx_Freeze* o *pyInstaller* presentan incompatibilidades con *Keras*, así que nos hemos visto obligados a incluir un entorno de Python 3.7 junto al código, y un pequeño script *batch* para Windows que lanza el `main.py` usando dicho entorno.



```
run_windows.bat X
1 start ./Python37/python.exe ./main.py
```

Ilustración 34: Contenido del script `run_windows.bat`

Como podemos observar en la Ilustración 34, el script que inicia la aplicación consta de una sola línea, que abre el fichero principal de código (`main.py`) en el intérprete de Python que incluimos. Pese a que el único acceso directo que ofrezcamos para ejecutar la herramienta es sólo compatible con Windows no implica que la aplicación en sí no sea multiplataforma, su funcionamiento es el correcto en MacOS y Linux, pero los usuarios de estos sistemas operativos deberán usar la terminal y ejecutar un comando equivalente al descrito en la ilustración válido para su sistema concreto. Sin embargo, esto no causa ningún inconveniente al equipo de digitalización de la empresa, ya que todos ellos usan Windows en sus equipos de trabajo y la solución provista les sirve.

9.4.2. Procesamiento y clasificación de las imágenes

El procesamiento y la clasificación de las imágenes reside en la clase AI. Una vez se da la orden de comenzar la generación del curso desde la interfaz gráfica, se inicializa una instancia de esta clase, y se comienzan a leer las imágenes (nombradas “DiapositivaX.JPG”, donde la X es el número de la diapositiva).

```
def read_images(self, imgsize):
    images = []
    paths = []

    for root, dirnames, filenames in os.walk(self.paths["images"].get()):
        self.log("Leyendo -> {} ".format(root))
        self.log("Número de ficheros: " + str(len(filenames)))
        for i in range(len(filenames)):
            path = os.path.join(root, "Diapositiva{}.JPG".format(str(i + 1)))
            img = cv2.imread(path, cv2.IMREAD_COLOR)
            try:
                img = cv2.resize(img, (imgsize, imgsize))
            except Exception as e:
                self.log("La imagen %s no se leyó correctamente. (¿Es una imagen?)" % path)
                self.log("Excepción: " + str(e))
            self.log("Leyendo --> " + path)
            paths.append(os.path.basename(path))
            images.append(np.array(img))

    self.log("Longitud del array de imágenes --> " + str(len(images)))
    return images, paths
```

Ilustración 35: Algoritmo de lectura de imágenes

Como podemos observar en la Ilustración 35, el proceso de lectura es sencillo, vamos iterando por cada uno de los ficheros y los vamos leyendo y redimensionando con *OpenCV*. También almacenamos información sobre las rutas de cada fichero en el array “*paths*”.

```

def predict_images(self, images, model, paths, use_gdt):
    if use_gdt:
        self.log("Usando nuevo modelo...")
        label_map = {'108': 0, '11': 1, '114': 2, '119': 3, '127': 4, '128': 5, '129': 6, '131': 7, '132': 8,
                    '133': 9, '134': 10, '135': 11, '136': 12, '137': 13, '138': 14, '139': 15, '141': 16,
                    '142': 17, '143': 18, '144': 19, '145': 20, '146': 21, '150': 22, '153': 23, '154': 24,
                    '155': 25, '39': 26}
    else:
        self.log("Usando modelo antiguo...")
        label_map = {'100': 0, '101': 1, '102': 2, '103': 3, '104': 4, '105': 5, '106': 6, '107': 7, '108': 8,
                    '109': 9, '110': 10, '111': 11, '112': 12, '113': 13, '114': 14, '115': 15, '116': 16,
                    '117': 17, '118': 18, '119': 19, '12': 20, '120': 21, '121': 22, '122': 23, '123': 24,
                    '124': 25, '125': 26, '126': 27, '13': 28, '15': 29, '18': 30, '19': 31, '20': 32, '21': 33,
                    '23': 34, '25': 35, '26': 36, '29': 37, '30': 38, '35': 39, '44': 40, '45': 41, '47': 42,
                    '48': 43, '52': 44, '53': 45, '7': 46, '72': 47, '76': 48, '86': 49, '93': 50, '94': 51,
                    '95': 52, '96': 53, '98': 54, '99': 55}

    aux = 0
    results = []
    for i in images:
        x = np.expand_dims(i, axis=0)
        features = model.predict(x)
        [results.append(int(k)) for k, v in label_map.items() if v == np.argmax(features)]

        self.log(paths[aux] + " => " + str(results[aux]))
        aux = aux + 1

    self.log("Cantidad de resultados --> " + str(len(results)))
    return results

```

Ilustración 36: Algoritmo de clasificación de las imágenes

Tras haber leído correctamente las imágenes, el siguiente paso es clasificarlas a través de la función representada en la Ilustración 36. En primer lugar, establecemos el mapa de etiquetas correspondiente al modelo que estemos usando. Esta tarea nos es útil para saber a qué plantillas se refieren las predicciones que emite la red neuronal y generar el código correcto. Sin embargo, una desventaja de este método es que requiere su actualización manual cada vez que se crea una nueva plantilla. Entraremos más en detalle sobre este aspecto en la sección de trabajo futuro.

Tras esto, el proceso es similar al descrito en la lectura de imágenes: vamos iterando por cada una y las pasamos por la red neuronal, almacenando las predicciones en un array de resultados.

9.4.3. Generación del código usando expresiones regulares

La generación de los archivos finales con todo lo que conlleva (extracción de códigos de plantillas, y extracción de imágenes y textos del manual) es el apartado más complejo de la aplicación final. Para cada módulo creamos un archivo titulado "MODULOX.php" (donde X es el número del módulo), e insertamos ciertas líneas de código que se incluyen al principio de todos los módulos por igual. En el caso del primer módulo, también insertamos las primeras cuatro diapositivas, que también son invariables.

PASO 1: VISITAR LA INTERFAZ

Ir a la web de PowerPoint *online* y crear una cuenta gratuita en <https://office.live.com/start/powerpoint.aspx>



Ilustración 37: La plantilla "126" ocupa varias diapositivas en los manuales, pero se puede representar en el código en una sola instancia de la plantilla

Así pues, el proceso es el siguiente: por cada módulo, vamos insertando una cantidad de plantillas igual al número de diapositivas de estos, omitiendo las repeticiones de las plantillas que se representan como una sucesión de diapositivas en los manuales pero que se pueden realizar con una sola instancia de la plantilla en el código (estas plantillas se encuentran en un array titulado "omit_repeating_diapos"). Podemos observar un ejemplo de este tipo de plantillas en la Ilustración 37.

```
def get_code_slide(self, target, slider):
    # Obteniendo resultado del Slider.php
    aux = "new Slider(" + str(target) + ","
    index = slider.find(aux)
    end_index = slider[index:].find(",")
    string = ""
    aux = re.search("<?php(.|\n)*?>", slider[index:index + end_index]).group()
    lines = aux.splitlines()
    for k, line in enumerate(lines):
        if k in {0, len(lines) - 1}:
            string += line.lstrip() + "\n"
        else:
            string += " " + line.lstrip() + "\n"
    return string
```

Ilustración 38: Algoritmo de extracción del código de las plantillas

```

"4" => new Slider(4,
    '<!-- ||| COMIENZO DIAPOSITIVA n ||| -->'
    '<?php
    $N++;
    ${diapo_.$N._img} = $img_rute."diapo10.jpg";
    ${diapo_.$N._h2} = "CÓMO CREAR UNA CAMPAÑA DE CONTENIDO PATROCINADO EN LINKEDIN ADS";
    ${diapo_.$N._p} = array();
    $filedata = file_get_contents($repo_rute."diapo4.txt");
    eval(">$filedata");
    <?
    <!-- ||| FINAL DIAPOSITIVA n ||| -->',
    'curso_md_gc_4.jpg'),
"5" => new Slider(5,
    '<!-- ||| COMIENZO diapositiva n ||| -->'

```

Ilustración 39: Estructura de las plantillas en el repositorio

La extracción del código de las plantillas se realiza a través de la función representada en la Ilustración 38, y en la Ilustración 39 se muestra un ejemplo de cómo se estructuran éstas en el repositorio (concretamente el fichero `model/slider.php`). En dicho fichero, los códigos de todas las plantillas disponibles se encuentran almacenadas de forma secuencial indexadas por su identificador. Así pues, para extraer el código de las plantillas que buscamos realizamos los siguientes pasos:

1. Situamos un puntero (index) al inicio de la plantilla que nos interesa buscando el texto "new Slider(" + identificador + ",".
2. Situamos otro puntero (end_index) al final de la plantilla que nos interesa buscando la primera instancia del texto "," desde el puntero de inicio. Tras esto, entre los dos punteros se encontrará el código de una sola plantilla, que es la que buscamos.
3. Seleccionamos el código de la plantilla con la expresión regular `<\?php(.|\n)*\?>`
4. Lo descomponemos por líneas y agregamos cada línea al resultado, recortando los espacios en blanco que las preceden (para tener el control del sangrado al insertarlas en los ficheros que generemos)
5. Devolvemos el resultado.

9.4.4. Extracción y procesamiento de las imágenes del manual

La extracción y el procesamiento de las imágenes del manual se realiza en el fichero `ppt.py` usando la librería `python-pptx`. Cada vez que generamos el fichero de código de un módulo, llamamos al método "create_images", que extrae las imágenes del manual pasándole entre otros parámetros los índices de las diapositivas de inicio y fin del módulo.

```

def slide_pictures_iterator(self, first_slide, last_slide):
    self.slide_number = 0
    for slide in self.ppt.slides:
        if self.ppt.slides.index(slide) in range(first_slide, last_slide):
            self.slide_number = self.slide_number + 1
            self.image_number = 0
            for shape in slide.shapes:
                if shape.shape_type == MSO_SHAPE_TYPE.PICTURE:
                    yield shape

```

Ilustración 40: Iterador de imágenes en cada diapositiva

Para el correcto funcionamiento de dicho método, es necesario el uso de la función auxiliar mostrada en la Ilustración 40, que otorga un iterador de las imágenes que se encuentran en cada una de las diapositivas que comprende el módulo.

```
def create_images(self, dir_path, first_slide, last_slide, results, convert_to_grayscale, ppt_images):
    self.image_hashes = {}
    for picture in self.slide_pictures_iterator(first_slide, last_slide):
        self.image_number = self.image_number + 1
        image = picture.image
        image_bytes = image.blob
        image_filename = str(dir_path) + str(self.slide_number) + "_" + str(self.image_number) + '.' + image.ext
        if image.shal not in self.image_hashes:
            self.image_hashes[image.shal] = os.path.basename(image_filename)
            with open(image_filename, 'wb') as f:
                f.write(image_bytes)
                if results[first_slide + self.slide_number - 1] in self.diapos_in_color or not convert_to_grayscale:
                    try:
                        image_pil = Image.open(image_filename)
                        print("Dejando en color: " + image_filename + " => (Diapo",
                              str(first_slide + self.slide_number - 1) + ")",
                              results[first_slide + self.slide_number - 1])
                    except:
                        print("Error al procesar imagen: " + image_filename + " => (Diapo",
                              str(first_slide + self.slide_number - 1) + ")",
                              results[first_slide + self.slide_number - 1])
                else:
                    try:
                        print("Convirtiendo a escala de grises: " + image_filename + " => (Diapo",
                              str(first_slide + self.slide_number - 1) + ")",
                              results[first_slide + self.slide_number - 1])
                        if image.ext == 'jpg':
                            image_pil = Image.open(image_filename).convert('L')
                        if image.ext == 'png':
                            image_pil = Image.open(image_filename).convert('LA')
                    except:
                        print("Error al convertir a escala de grises: " + image_filename + " => (Diapo",
                              str(first_slide + self.slide_number - 1) + ")",
                              results[first_slide + self.slide_number - 1])

                image_pil.thumbnail((1200, 1200))
                image_pil.save(image_filename)
                ppt_images[first_slide + self.slide_number - 1].append(os.path.basename(image_filename))
        else:
            ppt_images[first_slide + self.slide_number - 1].append(self.image_hashes[image.shal])
```

Ilustración 41: Algoritmo de extracción de las imágenes

Así pues, y como podemos observar en la Ilustración 41, por cada una de las imágenes que nos otorga el iterador la abrimos con la librería *Pillow* en color o en escala de grises dependiendo de la elección del usuario. Sin embargo, hay algunas plantillas cuyas imágenes siempre van en color, que son las que se encuentran en el array “*diapos_in_color*”. Si la imagen se procesó correctamente, la guardamos en un nuevo fichero con un nombre que le generamos previamente.

A veces, una misma imagen aparece varias veces en distintas diapositivas, y para evitar generar más de un archivo para la misma imagen y favorecer el ahorro de espacio disponemos de un array de *hashes* de las imágenes que vamos procesando. Por cada imagen que nos otorga el iterador, si su *hash* aparece en el array de *hashes* mencionado, la saltamos. También almacenamos las rutas de las imágenes generadas en otro array, lo que nos resulta útil para insertarlas en el código generado y asistir al programador a saber en qué diapositiva debe ir cada imagen.

```

def add_images_to_code(self, string, ppt_images, current_diapo_in_ppt):
    if re.search(r'\${diapo_\.\$N\._(background_image|img)} = \$\+', string) is not None and ppt_images[current_diapo_in_ppt + 1]:
        string = re.sub(r'\${diapo_\.\$N\._background_image} = \$\+',
                        '\${diapo_\.\$N\._background_image} = $img_rute.$N."' + ppt_images[current_diapo_in_ppt + 1][
                            0] + "';",
                        string)
        string = re.sub(r'\${diapo_\.\$N\._img} = \$\+',
                        '\${diapo_\.\$N\._img} = $img_rute.$N."' + ppt_images[current_diapo_in_ppt + 1][0] + "';",
                        string)
    fotos_diapo = ppt_images[current_diapo_in_ppt + 1]
    return string, fotos_diapo

```

Ilustración 42: Método para la inserción de las rutas de las imágenes en el código generado

Por lo general, las rutas de las imágenes se insertarán en el mismo sitio que los textos encontrados en el manual, y depende del usuario si los prefiere en los mismos ficheros de código generados o en ficheros aparte. Sin embargo, hay otra funcionalidad implementada que facilita en gran medida la calidad de vida del programador mientras digitaliza cursos: por cada plantilla insertada, si sigue la nomenclatura estándar en las variables que contienen las rutas de las imágenes, no hará falta que el usuario las inserte manualmente, gracias al método que podemos observar en la Ilustración 42.

Habitualmente, si una plantilla tiene imágenes, las variables que contengan las rutas de estas tendrán el nombre *background_image* o *img*. Así pues, por cada plantilla insertada comprobamos a través de una expresión regular si tiene alguna de las dos variables y si se extrajo alguna imagen en la diapositiva del manual correspondiente, y si ambas condiciones resultan verdaderas insertamos automáticamente la ruta de la imagen en el lugar correcto, ahorrándole unos segundos más de trabajo al programador.

9.4.5. Extracción y procesamiento de los textos del manual

La extracción y el procesamiento de los textos sigue una estructura similar a la descrita anteriormente para las imágenes. En este caso no necesitamos definir un iterador personalizado ya que podemos simplemente iterar a través de las diapositivas del manual.

```

def read_text_from_slides(self):
    text_total = []
    for slide in self.ppt.slides:
        slide_text = []
        for shape in slide.shapes:
            if not shape.has_text_frame or not len(shape.text) > 1:
                continue
            text_frame = shape.text_frame
            shape_text = ""
            for paragraph in text_frame.paragraphs:
                for run in paragraph.runs:
                    run_txt = run.text
                    if not len(run_txt) > 1:
                        continue
                    run_txt = re.sub(r'(<![/\ \w]+>|<!--|-->)', r''.htmlspecialchars("\g<0>").'', run_txt)
                    if run.font.italic:
                        run_txt = "<i>" + run_txt + "</i>"
                    if run.font.bold:
                        run_txt = "<bold_p>" + run_txt + "</bold_p>"
                    shape_text = shape_text + run_txt
                shape_text = shape_text + "<br>"
            shape_text = re.sub(r'(?:(?:https?|ftp):\/\/)?[\w\/-?=#.]+\.[a-z\/-?=#.]+\.??',
                               r'<a href="\g<0>" class="hyperlink" target="_blank">\g<0></a>',
                               shape_text)[0:-4]
            slide_text.append(shape_text)
        text_total.append(slide_text)

    return text_total

```

Ilustración 43: Algoritmo de extracción de los textos

Como podemos observar en la Ilustración 43, la librería *Python-pptx* nos otorga utilidades para iterar sobre cada una de las figuras que tiene cada diapositiva a través de la estructura “*shapes*”. Estas figuras pueden ser cuadros de texto, imágenes o gráficos, entre otros. Así pues, debemos asegurarnos de que cada figura que procesamos tiene texto dentro.

Además de la distinción entre figuras y diapositivas, dentro de los textos de las figuras existen los párrafos, y a su vez dentro de estos las “*runs*”, que nos referiremos a ellas como series. Una serie es un conjunto de texto dentro de un párrafo que mantiene un estilo constante (tamaño de letra, tipografía, cursiva, subrayado, etcétera).

Las series contienen variables que nos indican su estilo, y esto nos resulta bastante útil para permitir a la herramienta simplificar aún más el trabajo del programador mientras digitaliza cursos. Cuando se insertan los textos a mostrar en el código de las plantillas, es necesario insertar las etiquetas HTML correspondientes para indicar cuando el texto está en cursiva, negrita o es un enlace. Este proceso había que hacerlo a mano, observando cómo está estructurado el texto en el manual, y se podía convertir en una tarea bastante tediosa, ya que, por ejemplo, todas las palabras en inglés deben estar en cursiva, y este hecho podía dar lugar a diapositivas cuyos textos requerían una cantidad enorme de etiquetas. Sin embargo, como desde el código tenemos acceso a esta información, podemos extraer el texto incluyendo las etiquetas necesarias con asombrosa precisión.

9.4.6. Generación de los ficheros HTML para la creación de SCORM

La generación de los ficheros HTML para la creación de SCORM resulta bastante sencilla usando el módulo *requests* de Python:

```

def get_module(url):
    try:
        print("Haciendo petición a ", url)
        response = requests.get(url)
        return response.text.encode().decode('utf-8-sig')
    except:
        print("Fallo obtención de la URL {}, comprueba que está bien ".format(url))
        return -1

def create_file(name, path, text):
    with open(path+"/"+name, "wb") as f:
        print("Creando {} .... en {}".format(name, path))
        print(text.encode("utf-8"))
        f.write(text.encode("utf-8"))
        print("Texto introducido")
        f.close()

def create_html_files(url, number_of_modules, destination_path, fulp):
    for i in range(1, number_of_modules + 1):
        print('='*15)
        text = get_module(str(url + "MODULO{}.php").format(i))
        if text == -1:
            return -1
        name = "MODULO{}.html" if not fulp else "MODULO{}_FULP.html"
        create_file(
            name.format(i),
            destination_path,
            text
        )
        print('='*15)
    return 0

```

Ilustración 44: Métodos relacionados a la generación de los ficheros HTML para la conversión de los cursos a formato SCORM

Como podemos observar en la Ilustración 44, el proceso se reduce a hacer un número de peticiones HTTP igual al número de módulos del curso, y guardar las respuestas en archivos cuya nomenclatura elige el usuario previamente, correspondiendo a si el SCORM va a estar alojado en los servidores de la FULP o no.

9.4.7. Desarrollo de la interfaz de usuario

La interfaz gráfica de la aplicación está realizada con la librería *Tkinter*, y a continuación mostramos las interfaces finales de cada caso de uso:

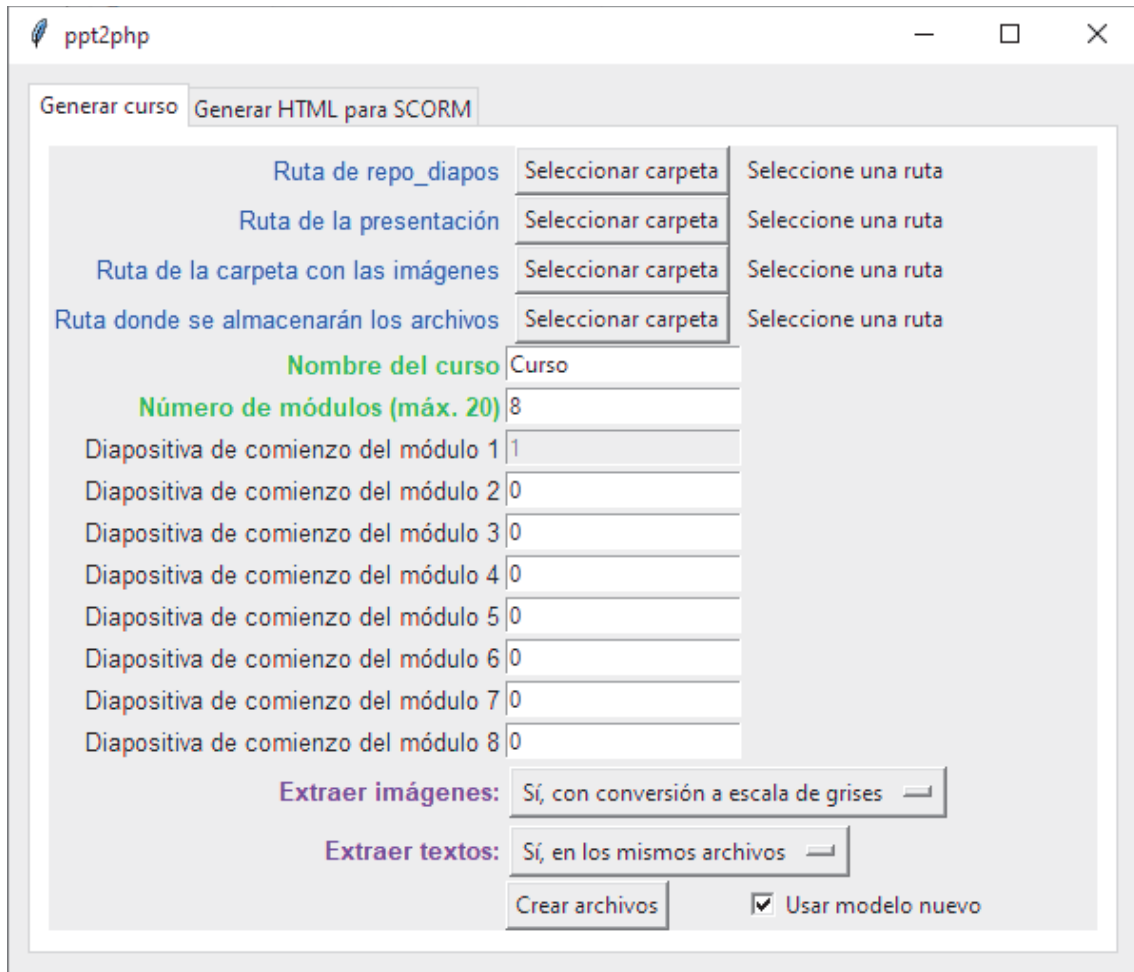


Ilustración 45: Interfaz gráfica del caso de uso "Generar curso"

Como podemos observar en la Ilustración 45, la interfaz final del caso de uso de "Generar curso" se asemeja en gran medida al mockup realizado y explicado anteriormente, conservando la estructura ideada y los detalles como la actualización automática del número de campos indicadores de las diapositivas de comienzo de cada módulo en relación con el número total de estos.

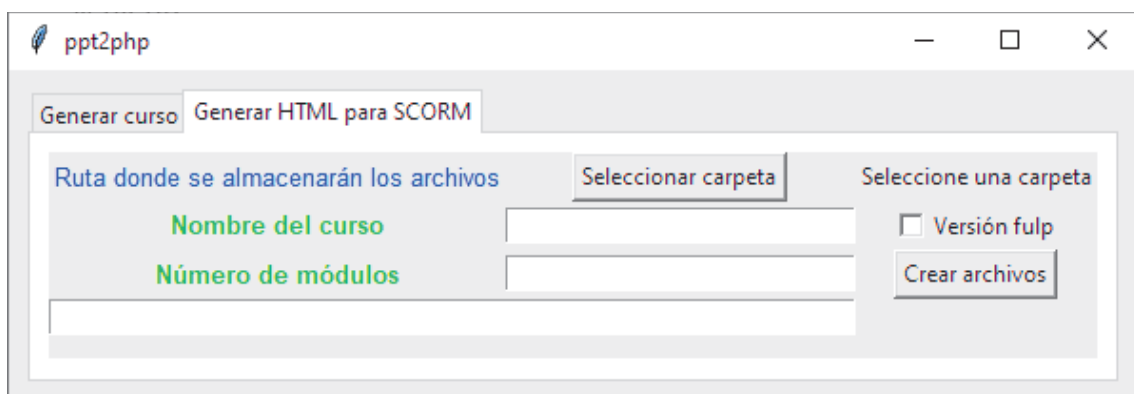


Ilustración 46: Interfaz gráfica del caso de uso "Generar HTML para SCORM"

Por otra parte, en la Ilustración 46 se representa el resultado de la interfaz del caso de uso de generación de ficheros HTML. Como podemos observar, esta interfaz también guarda una enorme semejanza al mockup mostrado anteriormente.

Para crear ambas interfaces, hacemos uso de los múltiples elementos altamente configurables que nos provee *Tkinter* (botones, campos de texto, etiquetas, pestañas, *checkboxes*, menús de opciones y demás). Además, cuando pulsamos los botones de “Seleccionar carpeta”, se abre una ventana con el explorador de Windows para buscar la ruta deseada, y la ruta seleccionada se muestra en la etiqueta a la derecha del botón, correspondiéndose al comportamiento descrito previamente.

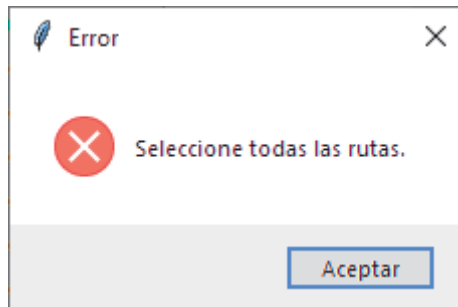


Ilustración 47: Ventana emergente con un mensaje de error al introducir las rutas incorrectamente durante la generación de un curso

También se dispone de prevención de errores: si falta alguna ruta o se ha introducido algún valor inválido, notificamos al usuario del error con una ventana emergente, como podemos observar en la Ilustración 47.

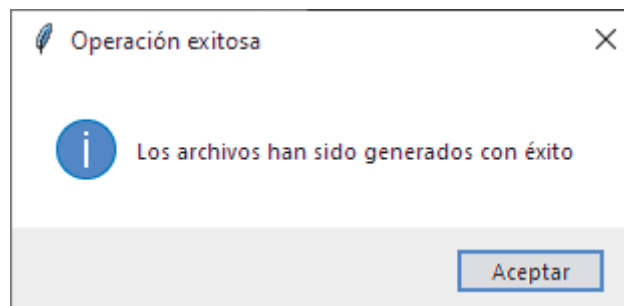


Ilustración 48: Ventana emergente con un mensaje de éxito al completar la operación

Por último, al terminar de generar los archivos, también se muestra una ventana emergente indicando al usuario del éxito de la operación, como podemos observar en la Ilustración 48.

10. Capítulo 10: Pruebas

10.1. Pruebas del modelo antiguo

Para probar el modelo de plantillas antiguo, hemos procesado el curso “Oficina sin papeles – La facturación electrónica” del programa Digitalizados de la FULP y hemos analizado los resultados comparando los resultados obtenidos con las plantillas reales que contiene el curso.

NÚMERO DE DIAPOSITIVA	PLANTILLA DETECTADA	PLANTILLA REAL	PREDICCIÓN CORRECTA
1	93	93	SÍ
2	94	94	SÍ
3	100	100	SÍ
4	118	118	SÍ
5	118	118	SÍ
6	118	118	SÍ
7	111	111	SÍ
8	94	94	SÍ
9	12	12	SÍ
10	12	12	SÍ
11	115	115	SÍ
12	102	102	SÍ
13	102	102	SÍ
14	102	102	SÍ
15	102	102	SÍ
16	115	115	SÍ
17	102	102	SÍ
18	102	102	SÍ
19	102	102	SÍ
20	115	115	SÍ
21	115	23	NO
22	12	12	SÍ
23	12	12	SÍ
24	12	12	SÍ
25	12	12	SÍ
26	52	52	SÍ
27	45	52	NO
28	94	94	SÍ
29	45	45	SÍ
30	118	118	SÍ
31	118	118	SÍ
32	118	118	SÍ
33	45	45	SÍ
34	118	118	SÍ
35	118	118	SÍ
36	118	118	SÍ
37	118	118	SÍ

38	45	45	SÍ
39	52	118	NO
40	118	118	SÍ
41	45	45	SÍ
42	118	118	SÍ
43	118	118	SÍ
44	118	118	SÍ
45	94	94	SÍ
46	52	52	SÍ
47	98	12	NO
48	118	118	SÍ
49	118	118	SÍ
50	118	118	SÍ
51	118	118	SÍ
52	52	12	NO
53	98	12	NO
54	12	52	NO
55	52	52	SÍ
56	98	12	NO
57	98	12	NO
58	118	118	SÍ
59	118	118	SÍ
60	118	118	SÍ
61	118	118	SÍ
62	52	52	SÍ
63	94	94	SÍ
64	108	108	SÍ
65	108	108	SÍ
66	108	108	SÍ
67	45	45	SÍ
68	99	12	NO
69	12	115	NO
70	122	122	SÍ
71	122	122	SÍ
72	122	122	SÍ
73	115	115	SÍ
74	45	45	SÍ
75	118	118	SÍ
76	118	118	SÍ
77	115	115	SÍ
78	122	122	SÍ
79	122	122	SÍ
80	122	122	SÍ
81	122	122	SÍ

82	118	115	NO
83	45	45	SÍ
84	118	118	SÍ
85	118	118	SÍ
86	52	115	NO
87	122	122	SÍ
88	122	122	SÍ
89	115	115	SÍ
90	45	45	SÍ
91	12	12	SÍ
92	115	115	SÍ
93	12	12	SÍ
94	115	115	SÍ
95	45	45	SÍ
96	118	118	SÍ
97	118	118	SÍ
98	115	115	SÍ
99	12	12	SÍ
100	115	115	SÍ
101	12	12	SÍ
102	45	45	SÍ
103	12	12	SÍ
104	12	115	NO
105	122	122	SÍ
106	122	122	SÍ
107	115	115	SÍ
108	45	45	SÍ
109	118	118	SÍ
110	118	118	SÍ
111	7	115	NO
112	122	122	SÍ
113	122	122	SÍ
114	45	45	SÍ
115	118	118	SÍ
116	12	118	NO
117	7	115	NO
118	122	122	SÍ
119	122	122	SÍ
120	45	45	SÍ
121	118	118	SÍ
122	118	118	SÍ
123	52	115	NO
124	122	122	SÍ
125	122	122	SÍ

126	118	115	NO
127	45	45	SÍ
128	118	118	SÍ
129	118	118	SÍ
130	12	115	NO
131	122	122	SÍ
132	122	122	SÍ
133	122	122	SÍ
134	12	115	NO
135	122	122	SÍ
136	122	122	SÍ
137	12	12	SÍ
138	45	45	SÍ
139	118	118	SÍ
140	118	118	SÍ
141	52	115	NO
142	122	122	SÍ
143	122	122	SÍ
144	12	12	SÍ
145	118	115	NO
146	45	45	SÍ
147	118	118	SÍ
148	118	118	SÍ
149	115	115	SÍ
150	122	122	SÍ
151	122	122	SÍ
152	115	115	SÍ
153	12	12	SÍ
154	12	12	SÍ
155	113	115	NO
156	45	45	SÍ
157	12	12	SÍ
158	115	115	SÍ
159	122	122	SÍ
160	122	122	SÍ
161	122	122	SÍ
162	122	122	SÍ
163	118	115	NO
164	45	45	SÍ
165	118	118	SÍ
166	118	118	SÍ
167	115	115	SÍ
168	122	122	SÍ
169	122	122	SÍ

170	122	122	SÍ
171	122	122	SÍ
172	115	115	SÍ
173	45	45	SÍ
174	12	12	SÍ
175	125	115	NO
176	122	122	SÍ
177	122	122	SÍ
178	122	122	SÍ
179	125	115	NO
180	122	122	SÍ
181	122	122	SÍ
182	125	115	NO
183	94	94	SÍ
184	108	108	SÍ
185	108	108	SÍ
186	108	108	SÍ
187	118	52	NO
188	52	52	SÍ
189	125	125	SÍ
190	125	125	SÍ
191	125	125	SÍ
192	125	125	SÍ
193	111	111	SÍ
194	45	45	SÍ

Tabla 4: Resultados de las clasificaciones por parte de la red neuronal frente a las plantillas reales, con el curso "Oficina sin papeles - La facturación electrónica", perteneciente al modelo antiguo

Como podemos observar en la Tabla 4, de las **194** diapositivas que contenía el curso, la herramienta ha detectado correctamente **165**, teniendo así pues **29** fallos y un **85,05%** de precisión. El recuento de predicciones es el siguiente:

Recuento de predicciones

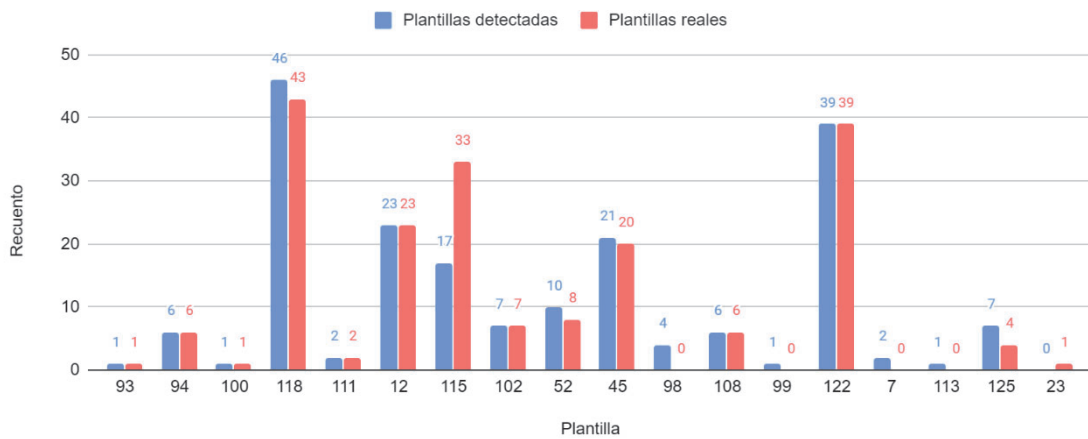


Ilustración 49: Recuento de las predicciones realizadas en el curso perteneciente al modelo antiguo

Mediante el gráfico representado en la Ilustración 49, podemos comprobar que las predicciones en general han sido acertadas, aunque ha habido ciertas dificultades al detectar la plantilla 115, detectándose aproximadamente la mitad de las veces que debería. También ha habido algunas detecciones de diapositivas que no aparecían en el curso (98, 99, 7, 113). Procedemos ahora a mostrar la distribución de errores.

Distribución de errores

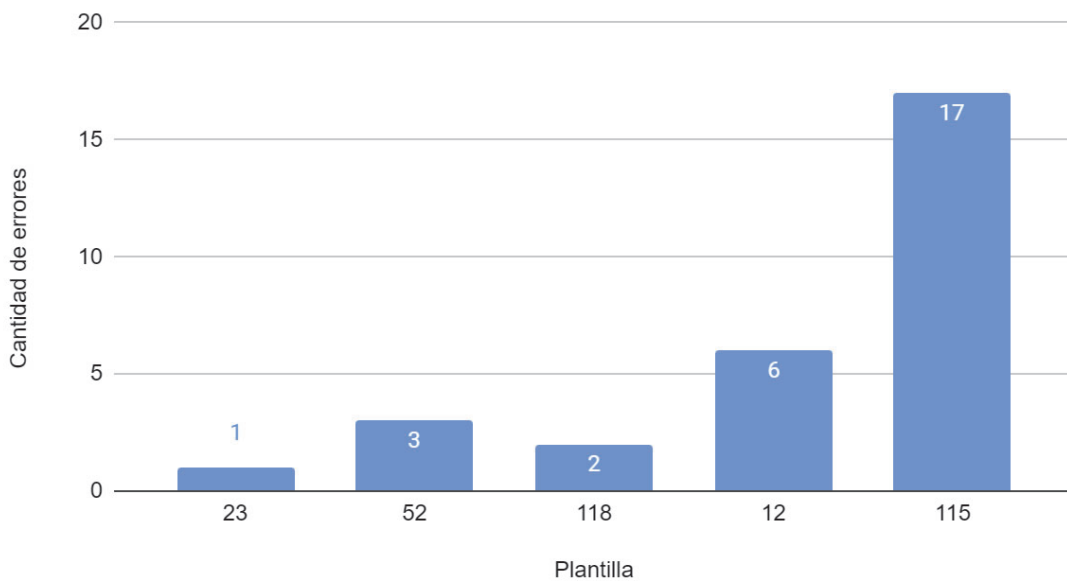


Ilustración 50: Distribución de los errores realizados en las clasificaciones realizadas en el curso perteneciente al modelo antiguo

Como habíamos mencionado anteriormente, y como podemos observar también en la Ilustración 50, la plantilla que causó más problemas a la red neuronal fue la 115, siendo la fuente de 17 de los 29 fallos en las predicciones. Por otra parte, la red también ha tenido problemas detectando la plantilla 12, y en menor medida, la 52, 118 y 23.

Pese a haber varias decenas de plantillas disponibles en el repositorio, cada curso se suele centrar en un subconjunto reducido de las mismas, para favorecer que los cursos se vean más distintos entre sí. Sin embargo, si se da el caso de que una de las plantillas que más se usa en un curso es difícil de detectar por la red, como se da en este caso, la precisión total en las predicciones disminuye notablemente.

De cualquier forma, pese a ocupar una sola plantilla más de la mitad de los errores, la precisión sigue resultando un ser más que notable 85%, que aumentará según vayamos mejorando la calidad de los conjuntos de datos que usamos para entrenar las redes. Un 85% de aciertos se aleja del 95% obtenido en el entrenamiento y el 94% obtenido en la validación, pero, como hemos mencionado anteriormente, es una precisión que permite a los programadores obtener un enorme aumento en rendimiento mientras digitalizan cursos.

10.2. Pruebas del modelo GDT

Las pruebas del modelo GDT siguen la misma estructura mostrada en las pruebas del modelo antiguo, esta vez con el curso “Capacidad de organización y planificación”. Comenzamos mostrando los resultados de las predicciones comparados con los datos reales.

NÚMERO DE DIAPOSITIVA	PLANTILLA DETECTADA	PLANTILLA REAL	PREDICCIÓN CORRECTA
1	128	128	SÍ
2	129	129	SÍ
3	141	141	SÍ
4	153	153	SÍ
5	136	136	SÍ
6	154	154	SÍ
7	154	154	SÍ
8	146	146	SÍ
9	127	127	SÍ
10	108	108	SÍ
11	108	108	SÍ
12	108	108	SÍ
13	154	154	SÍ
14	155	155	SÍ
15	154	154	SÍ
16	129	129	SÍ
17	132	132	SÍ
18	132	132	SÍ
19	132	132	SÍ
20	132	132	SÍ
21	132	132	SÍ
22	132	132	SÍ
23	132	132	SÍ

24	154	154	SÍ
25	141	141	SÍ
26	129	129	SÍ
27	129	129	SÍ
28	137	137	SÍ
29	137	137	SÍ
30	137	137	SÍ
31	137	137	SÍ
32	137	137	SÍ
33	137	137	SÍ
34	137	137	SÍ
35	137	137	SÍ
36	154	154	SÍ
37	119	119	SÍ
38	119	119	SÍ
39	129	129	SÍ
40	154	154	SÍ
41	135	135	SÍ
42	154	154	SÍ
43	150	150	SÍ
44	134	134	SÍ
45	154	154	SÍ
46	129	129	SÍ
47	141	141	SÍ
48	129	129	SÍ
49	154	154	SÍ
50	11	11	SÍ
51	144	144	SÍ
52	154	154	SÍ
53	11	11	SÍ
54	114	146	NO
55	129	129	SÍ
56	154	39	NO
57	39	39	SÍ
58	114	114	SÍ
59	114	114	SÍ
60	114	114	SÍ
61	114	114	SÍ
62	114	114	SÍ
63	114	114	SÍ
64	39	39	SÍ
65	114	114	SÍ
66	114	114	SÍ
67	114	114	SÍ

68	114	114	SÍ
69	114	114	SÍ
70	114	114	SÍ
71	114	114	SÍ
72	129	129	SÍ
73	146	11	NO
74	108	108	SÍ
75	108	108	SÍ
76	108	108	SÍ
77	154	154	SÍ
78	135	135	SÍ
79	128	128	SÍ

Tabla 5: Resultados de las clasificaciones por parte de la red neuronal frente a las plantillas reales, con el curso "Capacidad de organización y organización", perteneciente al modelo GDT

Como podemos observar en la Tabla 5, de las **79** diapositivas que contenía el curso, la herramienta ha detectado correctamente **76**, teniendo así pues **3** fallos y un **96,2%** de precisión. El recuento de predicciones es el siguiente:

Recuento de predicciones

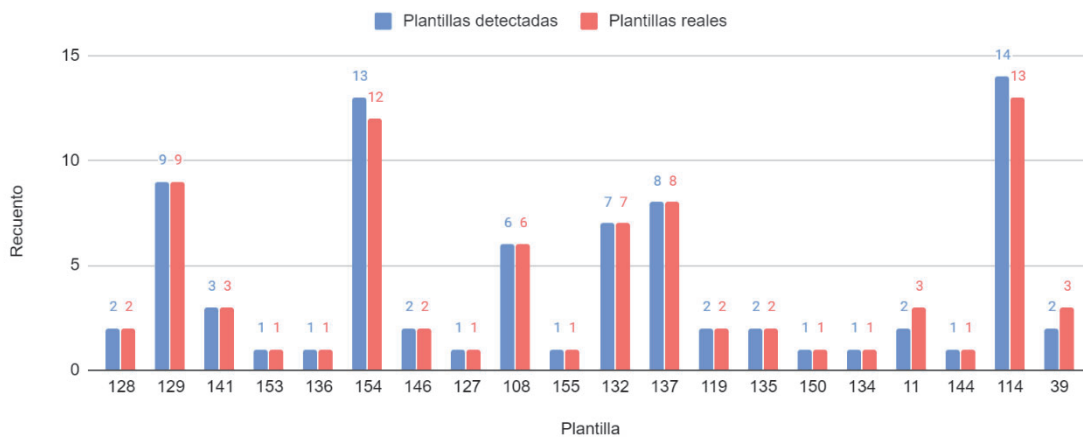


Ilustración 51: Recuento de las predicciones realizadas en el curso perteneciente al modelo GDT

Como podemos observar en la Ilustración 51, todas las predicciones han sido certeras excepto tres, concretamente en relación con las plantillas 154, 114 y 39. Este hecho también lo podremos observar en la distribución de errores:

Distribución de errores

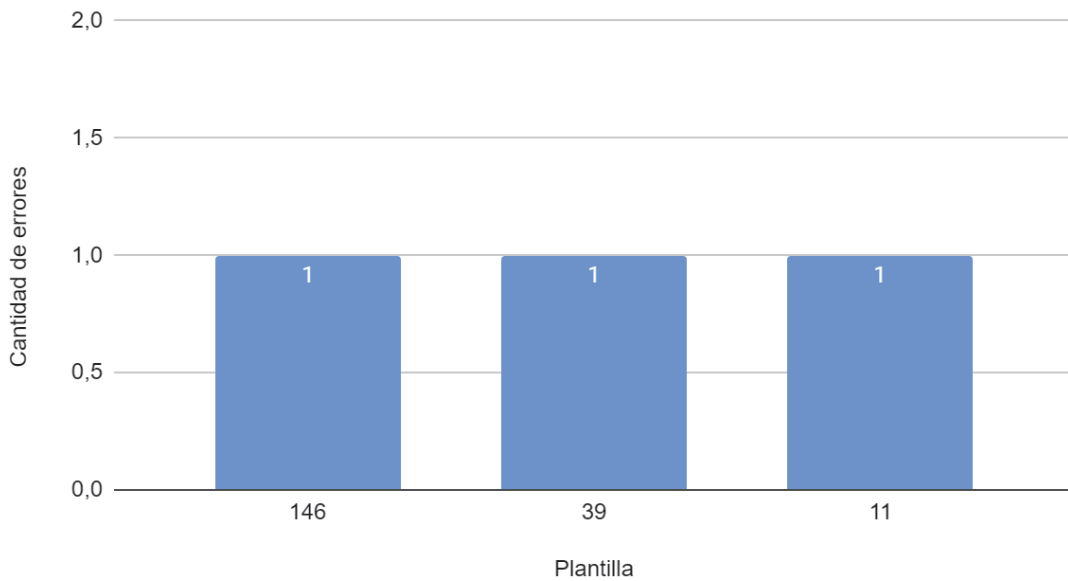


Ilustración 52: Distribución de los errores realizados en las clasificaciones realizadas en el curso perteneciente al modelo GDT

Sin duda, como podemos observar en el gráfico representado en la Ilustración 52, en este caso la herramienta ha sido mucho más precisa que con el curso anterior, incluso habiendo sido entrenada con un conjunto de datos mucho más reducido. Es más, la precisión en la fase de pruebas ha sido incluso mayor que la obtenida en la fase de entrenamiento.

Atribuimos este hecho a que, pese a que el conjunto de datos es más pequeño, el número de plantillas también es bastante menor, y están mucho más diferenciadas entre sí. En el modelo antiguo había varias plantillas que podían representar una amplia gama de diapositivas con apartados estilísticos muy variados, mientras que en este caso cada plantilla está más especializada.

Sin embargo, aunque esta precisión obtenida es muy positiva, el modelo GDT es aún joven y está en desarrollo, así que se deberá ir mejorando el conjunto de datos y reentrenando la red periódicamente para evitar que se desfase.

11. Capítulo 11: Resultados, conclusiones y trabajo futuro

11.1. Resultados y conclusiones

Tras concluir todas las fases del desarrollo, hemos conseguido una herramienta que consigue sus objetivos notablemente, resultando en un aumento de la productividad significativo. De esta forma, la herramienta se ha convertido en la navaja suiza de los programadores que integran el equipo de digitalización de cursos, gracias no solo a su generación de código

automática, sino también a la extracción, nombrado e inserción automática en el código de las imágenes, la extracción de los textos incluyendo las etiquetas HTML necesarias, y la generación de ficheros HTML para la conversión de los cursos a SCORM.

A nivel personal, el desarrollo de este Trabajo de Fin de Grado le ha resultado muy enriquecedor a su autor, ya que le ha permitido dar uso a sus conocimientos sobre *Machine Learning* e Inteligencia Artificial para crear un proyecto desde cero de ámbito profesional, que otorga valor real a la empresa en la que trabaja.

Desde que en la asignatura Fundamentos de los Sistemas Inteligentes se le impartieron al autor del presente trabajo los primeros conocimientos sobre redes neuronales le fascinó el sector, y continuó investigando de tal forma que realizó un programa especializado de *Deep Learning* compuesto por cinco cursos en Coursera que le favoreció mucho de cara a saber solucionar los problemas en el desarrollo del proyecto.

Las tareas que nos permite automatizar la herramienta desarrollada no presentan una elevada dificultad o complejidad, pero sí que requieren una cantidad excesiva de tiempo debido al gran volumen de diapositivas que tiene cada curso. Además, dichas tareas resultan enormemente tediosas y mecánicas, y eliminarlas permite a los programadores disfrutar más del proceso de digitalización de cursos y usar más frecuentemente sus habilidades técnicas en el resto de las tareas que sí las requieren en mayor medida.

Gracias a este abanico de funciones, nos podemos asegurar que, aunque las predicciones de las plantillas bajasen de precisión, la herramienta siempre será útil para los programadores de la empresa, ya que el resto de las funciones no fallan nunca. Además, a medida que sigamos implementando nuevas funcionalidades y más mejoras a la experiencia de uso, cada vez será menor el tiempo requerido para digitalizar un curso, lo que implicará una mayor cantidad de tiempo para pulir los detalles de cada uno, y mejorarlos de formas que antes no hubieran sido posibles.

El aumento de la velocidad de producción de cursos que esta herramienta provee es esencial para el éxito de [smartly.es](https://www.smartly.es), el portal de formación en línea que albergará todos los cursos que realiza la empresa. Mantener un flujo constante de nuevos cursos que aterricen en la plataforma servirá como el atrayente principal de nuevos clientes y aumentará las opciones de los ya existentes.

Sin duda, las Redes Neuronales y el *Machine Learning* tienen usos inimaginables. Desde su uso en la medicina para mejorar los diagnósticos, hasta su uso en los vehículos autónomos, la inteligencia artificial tiene el poder de cambiarnos la vida. Estamos rodeados de redes neuronales, desde las recomendaciones de contenido que se nos hacen en portales como Amazon que resultan ser gran parte de su éxito (Retta, s.f.), a los altavoces inteligentes como Google Home o Alexa que nos resuelven tareas comunes gracias al procesamiento y generación del lenguaje natural (Marr, 2018). Estas técnicas no tienen solo el poder de mejorar nuestras vidas personales, sino también de mejorar los procesos de trabajo de las empresas, como hemos demostrado en este proyecto.

El conocimiento profundo sobre el funcionamiento de estas herramientas y la capacidad de aplicarlas en problemas nuevos para solucionarlos es quizás una de las habilidades más cercanas a un superpoder que podemos adquirir en la actualidad.

11.2. Trabajo futuro

Existen múltiples funcionalidades que se implementarán con el paso del tiempo que mejorarán aún más la herramienta:

- Mejora de la calidad de los conjuntos de datos para la mejora de la precisión de las predicciones.
- Desarrollo de una API para que la aplicación final obtenga los mapas de etiquetas, las diapositivas que deben omitirse si se repiten y las que deben ir siempre en color.
- Una sección en la aplicación que permita a los programadores indicar qué diapositivas se detectaron incorrectamente, de manera que se agreguen automáticamente a los conjuntos de datos.
- Alojar los modelos de las redes neuronales en el servidor para reducir significativamente el peso de la herramienta (cada uno de los dos modelos ocupa aproximadamente 300MB)
- Automatización de la bibliografía y los exámenes finales de los cursos, que también se basan en código PHP.
- Generación e inserción automática de las dinámicas (minijuegos interactivos) en los cursos.
- Implementar un sistema de detección automática del número de módulos en los manuales y los números de las diapositivas en los que empieza cada uno para que el usuario no tenga que introducirlos al generar los cursos.

12. Capítulo 12: Bibliografía

Las fuentes bibliográficas usadas para la confección de esta memoria son las siguientes:

Apache License. (s.f.). Obtenido de apache.org: <https://www.apache.org/licenses/LICENSE-2.0>

Cartas, A. (16 de Abril de 2020). *Diagrama de un perceptrón con cinco señales de entrada*.

Obtenido de Wikipedia: <https://es.wikipedia.org/wiki/Perceptr%C3%B3n>

Chatterjee, C. C. (31 de Julio de 2019). *Basics of the Classic CNN*. Obtenido de towards data science: <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add>

Chris. (15 de Octubre de 2019). *Leaky ReLU: improving traditional ReLU*. Obtenido de Machine Curve: <https://www.machinecurve.com/index.php/2019/10/15/leaky-relu-improving-traditional-relu/>

- Dan Clark, K. (Mayo de 2018). *KDnuggets*. Obtenido de 7 Useful Suggestions from Andrew Ng “Machine Learning Yearning”: <https://www.kdnuggets.com/2018/05/7-useful-suggestions-machine-learning-yearning.html>
- GNU General Public License*. (s.f.). Obtenido de <https://www.gnu.org/licenses/gpl-3.0.html>
- Guerra, C. (s.f.). *El perceptrón*. Obtenido de <https://nbviewer.jupyter.org/url/cayetanoguerra.github.io/ia/nbpy/learning-perceptron.ipynb>
- Guerra, C. (s.f.). *Redes neuronales 1*. Obtenido de <https://nbviewer.jupyter.org/url/cayetanoguerra.github.io/ia/nbpy/redneuron1.ipynb>
- Luashchuk, A. (s.f.). *8 Reasons Why Python is Good for Artificial Intelligence and Machine Learning*. Obtenido de djangostars: <https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/>
- Marr, B. (12 de Enero de 2018). *As Google Home Fights Amazon Echo We Ask: What Really Is Natural Language Generation and Processing?* Obtenido de Forbes: <https://www.forbes.com/sites/bernardmarr/2018/01/12/as-google-home-fights-amazon-echo-we-ask-what-really-is-natural-language-generation-and-processing/>
- Moawad, A. (1 de Febrero de 2018). *Neural networks and back-propagation explained in a simple way*. Obtenido de Medium: <https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e>
- Retta, L. (s.f.). *Two decades in the making – the powerful strategies behind the Amazon recommendation engine*. Obtenido de Dynamic Yield: <https://www.dynamicyield.com/blog/amazon-recommendations/>
- Sharma, S. (6 de Septiembre de 2017). *Activation Functions in Neural Networks*. Obtenido de towards data science: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>
- StarUML 3*. (s.f.). Obtenido de StarUML 3: <http://staruml.io/>
- Thakur, R. (6 de Agosto de 2019). *Step by step VGG16 implementation in Keras for beginners*. Obtenido de towards data science: <https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c>
- The MIT License*. (s.f.). Obtenido de <https://opensource.org/licenses/MIT>
- Toro, L. (s.f.). *Jupyter notebook: documenta y ejecuta código desde el navegador*. Obtenido de Desde Linux: <https://blog.desdelinux.net/jupyter-notebook/>

ppt2php

Convierte manuales en código automáticamente gracias al Machine Learning

Material necesario:

- **Una carpeta con las imágenes que corresponden a las diapositivas del manual.** (Las diapositivas deben llamarse "DiapositivaX.JPG", siendo X el número de la diapositiva. PowerPoint provee esto muy fácilmente, se explica a continuación).
- **El manual del curso** en formato pptx.
- **ppt2php.zip**
- **repo_diapos** en una carpeta local.

Convertir un manual a código

Pasos que realizar:

1. Si ya se dispone de la carpeta con las imágenes del manual, salta al paso 2. Si no, realiza los siguientes pasos:
 1. Abre el manual en **Microsoft PowerPoint**.
 2. Selecciona **Guardar como**.
 3. Elige la ruta donde deseas la carpeta con las imágenes, selecciona como tipo de archivo **JPG** y presiona **Aceptar**.
 4. Selecciona **Todas las diapositivas**.
 5. En unos pocos segundos estará listo (dependiendo del tamaño del manual).
2. Descomprime **ppt2php.zip** y posteriormente ejecuta **run_windows.bat**. El programa se iniciará.
3. En la interfaz del programa, selecciona la ruta de la carpeta en la que se encuentra **repo_diapos** localmente, la ruta de la **carpeta con las imágenes**, la ruta del **manual del curso** y la ruta en la que deseas que se generen los archivos (el programa los almacenará en una nueva carpeta). **IMPORTANTE: Las rutas no pueden contener tildes.**
4. **(OPCIONAL)** Introduce el nombre del curso (determina el nombre de la carpeta que el programa creará).
5. Introduce el **número de módulos** que tiene el manual, y posteriormente los **números de las diapositivas donde empiezan cada uno de los módulos del manual**.
6. Escoge si deseas que se **extraigan las imágenes** y, en caso de que sí, si convirtiéndolas a **escala de grises** o **dejándolas en color**. Si se extraen, se generarán en una carpeta propia, divididas por módulos.
7. Escoge si deseas que se **extraigan los textos** y, en caso de que sí, si los deseas **en los mismos archivos de código generados** o en **ficheros aparte**.
8. Escoge si el curso usa el modelo nuevo de gestión del tiempo o el antiguo.
9. Presiona **Crear archivos**. Al finalizar, el programa te notificará. Los archivos generados se encontrarán en la ruta especificada previamente.

Generar curso Generar HTML para SCORM

Ruta de repo_diaos Seleccione una ruta

Ruta de la presentación Seleccione una ruta

Ruta de la carpeta con las imágenes Seleccione una ruta

Ruta donde se almacenarán los archivos Seleccione una ruta

Nombre del curso

Número de módulos (máx. 20)

Diapositiva de comienzo del módulo 1

Diapositiva de comienzo del módulo 2

Diapositiva de comienzo del módulo 3

Diapositiva de comienzo del módulo 4

Diapositiva de comienzo del módulo 5

Diapositiva de comienzo del módulo 6

Diapositiva de comienzo del módulo 7

Diapositiva de comienzo del módulo 8

Extraer imágenes:

Extraer textos:

Usar modelo nuevo

Generar los archivos HTML para generar el SCORM

Pasos que realizar:

1. Acceder a la pestaña “**Generar HTML para SCORM**”
2. Introducir el **nombre del curso** (el nombre interno en nuestros servidores).
3. Introducir el **número de módulos**.
4. Seleccionar la **ruta donde deseas los archivos**.
5. Escoge si los ficheros se nombrarán con el sufijo FULP o no (**versión FULP**).
6. (**OPCIONAL**) Modifica directamente la URL si deseas hacer peticiones a otra ruta
7. Presiona **Crear archivos**. Al finalizar, el programa te notificará. Los archivos generados se encontrarán en la ruta especificada previamente.

Generar curso Generar HTML para SCORM

Ruta donde se almacenarán los archivos Seleccione una carpeta

Nombre del curso

Número de módulos

Versión fulp