

REPORTE

CLASIFICADOR DE MARISCOS POR MEDIO DE ALGORITMOS BASADOS EN INTELIGENCIA ARTIFICIAL

QUE PARA OBTENER EL TÍTULO DE:

TÉCNICO SUPERIOR UNIVERSITARIO

EN MECATRÓNICA, ÁREA AUTOMATIZACIÓN

PRESENTA:

LÓPEZ LÓPEZ ISAAC ISAÍAS

AUTORIZACIÓN DEL
ASESOR DE EMPRESA

**DR. MARTÍN MORENO
GUZMÁN**

AUTORIZACIÓN DEL
ASESOR DE LA UTSJR

**DR. MARTÍN MORENO
GUZMÁN**

SAN JUAN DEL RÍO, QRO.

AGOSTO DE 2021

“La mejor Universidad para los mejores alumnos”



AGRADECIMIENTOS

Quiero agradecer enormemente a muchas personas y a Dios porque han estado conmigo durante mi trayectoria académica universitaria y mi vida:

A mi Madre, Mónica; y a mi hermana, Ximena:

Por su amor, su paciencia y por siempre estar conmigo en todo lo que me propongo.

A mis Madrinas y Padrinos:

Por brindarme su cariño y su gran apoyo a lo largo de todos estos años.

A mi Asesor de Estadía, Dr. Martín Moreno Guzmán:

Por brindarme la oportunidad de realizar mi proyecto con él y de desenvolverme en esta área que va de la mano con Mecatrónica y que sea ha vuelto mi área de interés dentro de la carrera; y por su gran paciencia y tiempo de espera para el desarrollo de este.

Al M. en A.P.I. Fidencio Díaz Méndez:

Por brindarme siempre su apoyo, desde los propedéuticos y hasta el día de hoy en todo lo que acontece en la Universidad.

Al M. en A. Rafael Ocampo Martínez:

Por ser mi primer tutor, por preocuparse desde el primer día por mi desempeño académico y mi vida estudiantil; y por sus consejos a lo largo de este tiempo.

Al Dr. Iván Trejo Zúñiga:

Por su apoyo en la carrera, aunque jamás me haya dada alguna clase y por ayudarme a enfocar una de mis pasiones, las matemáticas, dentro del ámbito ingenieril.

A Marius “Morriquitín” Gavroche:

Por ser un miembro más de la familia, brindarnos su compañía y enseñarnos a que siempre hay que ser felices.

A Sergio “Checo” Pérez:

Por ser una gran fuente de inspiración para mí y por mostrarme que siempre hay que dar lo mejor de uno mismo sin importar en la situación en la que uno se encuentre.

ÍNDICE

ÍNDICE DE FIGURAS	10
INTRODUCCIÓN.....	14
CAPÍTULO 1 ANTECEDENTES DE LA UNIVERSIDAD TECNOLÓGICA DE SAN JUAN DEL RÍO	15
1.1 Historia.....	15
1.2 Misión	15
1.3 Visión.....	15
1.4 Valores	16
1.5 Modelo Educativo.....	16
1.5.1 Intensidad	16
1.5.2 Continuidad	16
1.5.3 Calidad	16
1.5.4 Pertinencia	16
1.5.5 Polivalencia	17
1.5.6 Flexibilidad.....	17
1.6 Certificados y Premios	17
1.7 Área de Desarrollo	17
CAPÍTULO 2 GENERALIDADES DEL PROYECTO	18
2.1 Nombre del Proyecto	18
2.2 Problema	18
2.3 Objetivo General.....	18
2.4 Metodología	18
2.5 Alcance	19

CAPÍTULO 3 MARCO TEÓRICO	20
3.1 Inteligencia Artificial (IA)	20
3.1.1 Machine Learning (ML).....	21
3.1.2 Deep Learning (DL).....	21
3.2 Deep Learning.....	21
3.2.1 Redes Neuronales Artificiales (ANNs)	21
3.2.2 Redes Neuronales Convolucionales (CNNs)	22
3.2.3 Arquitecturas ConvNet.....	29
3.2.4 Consideraciones Computacionales.....	31
3.3 Procesamiento de Imágenes	33
3.3.1 Visión Computacional.....	33
3.3.2 Clasificación de Imágenes.....	34
3.3.2.1 Clasificadores Supervisados	36
3.3.2.2 Clasificadores No Supervisados	37
3.3.3 Segmentación de Imágenes	38
3.3.3.1 Discontinuidades	40
3.3.3.2 Similitud	43
3.4 Python	47
3.4.1 Numpy	49
3.4.2 Pandas.....	49
3.4.3 Scikit-Learn	50
3.4.4 Matplotlib	50
3.4.5 Seaborn.....	51
3.4.6 Os	51

3.4.7 Glob.....	51
3.4.8 Pathlib.....	51
3.4.9 OpenCV.....	52
3.4.10 TensorFlow.....	52
3.4.11 Keras.....	53
3.4.12 PyTorch	54
3.4.13 Otros Frameworks	54
3.5 Herramientas	55
3.5.1 Kaggle	55
3.5.2 Jupyter Notebook	56
3.5.3 Google Colaboratory	57
3.6 Microsoft Azure	58
3.6.1 Azure Portal.....	60
3.6.2 Azure Marketplace	60
3.6.3 Microsoft Cognitive Services.....	61
3.6.4 Custom Vision.....	62
3.7 GitHub	63
3.7.1 Git.....	65
CAPÍTULO 4 DESARROLLO DEL PROYECTO	66
4.1 Introducción al Entorno de Trabajo	66
4.1.1 Introducción a Google Colaboratory	67
4.1.2 Introducción a Kaggle	70
4.2 Principios de Python	72
4.2.1 Principios de NumPy.....	80

4.2.2 Principios de Manipulación de Datos con Pandas	81
4.2.3 Principios de Visualización con Matplotlib	83
4.3 Principios de Deep Learning.....	84
4.3.1 Introducción al Deep Learning.....	85
4.3.2 Redes Neuronales	86
4.3.3 Redes Neuronales Convolucionales	87
4.3.4 Redes Neuronales Recurrentes.....	88
4.3.5 Redes Generativas Antagónicas	89
4.3.6 Implementación de Modelos de Aprendizaje Automático.....	90
4.3.7 Recursos Complementarios.....	91
4.4 Principios de Git y GitHub	91
4.4.1 Instalación de Git y Git Bash	92
4.4.2 Creación del Repositorio Local e Identificación	98
4.4.3 Creación del Repositorio Remoto en GitHub	101
4.4.4 Conexión entre los repositorios local y GitHub	103
4.5 CNN para un dataset de mariscos	106
4.5.1 Clonación del Repositorio de GitHub en Colab.....	107
4.5.2 Importación de Librerías	107
4.5.3 Manipulación de Datos.....	108
4.5.3.1 Obteniendo Datos	108
4.5.3.2 Generación de Series	110
4.5.3.3 Generación del DataFrame	111
4.5.3.4 Mostrando Datos	112
4.5.3.5 Contando los Datos.....	114

4.5.4 Implementación del Modelo.....	114
4.5.4.1 División de los Datos.....	115
4.5.4.2 Generación de Datos.....	116
4.5.5 Desarrollo del Modelo.....	119
4.5.5.1 Arquitectura del Modelo.....	119
4.5.5.2 Resumen del Modelo	120
4.5.5.3 Compilado del Modelo	124
4.5.5.4 Entrenamiento del Modelo	124
4.5.6 Código en Kaggle.....	125
4.6 Principios de Azure y Custom Vision.....	128
4.6.1 Creación del Proyecto en Custom Vision Service	128
4.6.2 Carga de Imágenes Etiquetadas.....	131
4.6.3 Entrenamiento del Modelo	134
CAPÍTULO 5 PRUEBAS Y RESULTADOS	136
5.1 Resultados del Modelo CNN	136
5.1.1 Resultados del Entrenamiento	136
5.1.2 Evaluación del Modelo.....	137
5.1.3 Gráficas de Pérdida y Precisión	138
5.1.4 Reporte de Clasificación y Matriz de Confusión	140
5.2 Resultados del Modelo con Custom Vision Service.....	143
5.2.1 Resultados del Entrenamiento	143
5.2.2 Prueba del Modelo	144
5.2.2.1 Prueba Rápida (Quick Test)	144
5.2.2.2 Azure Cloud Shell	145

5.3 Diferencias entre los Modelos	149
5.4 Guardar un Notebook en GitHub.....	149
CONCLUSIONES.....	152
REFERENCIAS	153

ÍNDICE DE FIGURAS

Imagen 3.1 Estructura de una ANN con varias capas. Tomado de Kumar, Upadhyay, & Kumar (2020)	22
Imagen 3.2 Estructura básica de una CNN. Tomado de Borra, Thanki, & Dey (2019).....	22
Imagen 3.3 Diagrama de una operación convolucional 2D en una imagen y su resultado.....	23
Imagen 3.4 Arquitectura de una CNN multicapa. Tomado de Kumar, Upadhyay, & Kumar (2020).....	24
Imagen 3.5 Ejemplo de operación de filtrado. Tomado de Shanmugamani (2018).....	25
Imagen 3.6 Imagen muestra a la que se le va a aplicar el kernel. Tomado de IBM Development	26
Imagen 3.7 Imagen del resultado de la aplicación de un solo pase del kernel. Tomado de IBM Development (2018).....	26
Imagen 3.8 Ejemplo en el que un volumen de entrada es agrupado con un kernel y una zancada de tamaño 2. Tomado de Stanford University (2021)	27
Imagen 3.9 Ejemplo de la operación de agrupación más común, max pooling, con una zancada de 2. Tomado de Stanford University.....	28
Imagen 3.10 Red convolucional para el reconocimiento de dígitos escritos a mano. Tomado de (Gorner, 2021)	28
Imagen 3.11 Aprendizaje Residual, un componente. Tomado de He, Zhang, Ren, & Sun (2016).....	31
Imagen 3.12 Diagrama de un clasificador de imágenes basado en características. Tomado de Dey (2020)	35
Imagen 3.13 Diagrama de un clasificador de imágenes basado en Deep Learning. Tomado de Dey (2020)	35
Imagen 3.14 Clasificación Supervisada	36
Imagen 3.15 Clasificación No Supervisada.....	37
Imagen 3.16 Imagen muestra donde se aplicará la segmentación semántica.....	38
Imagen 3.17 Segmentación semántica de la Imagen 3.16. Tomado de Shanmugamani (2018)	39
Imagen 3.18 Segmentación de instancias donde cada instancia se distingue de una misma etiqueta (persona)	39
Imagen 3.19 (Sup. Izq.) Imagen original; (Sup. Der.) Ruido agregado; (Inf. Izq.) Salida del filtro; (Inf. Der.) Salida del umbral	41
Imagen 3.20 Imagen muestra con cuatro ejemplos de distintas salidas tras la aplicación de un distinto kernel en específico	42
Imagen 3.21 Detección de bordes por operadores de derivada: (a) Franja clara sobre un fondo oscuro; (b) Franja oscura sobre un fondo claro.....	43
Imagen 3.22 Histogramas de intensidad que pueden dividirse (a) por un único umbral; (b) por un umbral dual. Tomado de González & Woods (2018).....	44
Imagen 3.23 Resultado de un thresholding no adaptativo. Tomado de Chan	45
Imagen 3.24 (a) Imagen original con un punto de ‘semilla’; (b) Etapa temprana del Crecimiento de Región; (c) Región final. Tomado de Chan	45
Imagen 3.25 Imagen R divida con su respectivo quadtree. Tomado de González & Woods (2018).....	46
Imagen 2.26 Logo de NumPy	49
Imagen 3.27 Hoyo Negro M87. Tomado de Event Horizon Telescope (EHT) (2019).....	49
Imagen 3.28 Logo de pandas	50
Imagen 3.29 Logo de scikit-learn.....	50
Imagen 3.30 Logo de matplotlib	50
Imagen 3.31 Logo de seaborn	51
Imagen 3.32 Logo de OpenCV	52
Imagen 3.33 Logo de TensorFlow	53
Imagen 3.34 Logo de Keras	53
Imagen 3.35 Logo de PyTorch	54
Imagen 3.36 Sitio Web Kaggle en la sección de competiciones	56
Imagen 3.37 Interfaz de usuario de un documento de Jupyter Notebook	56
Imagen 3.38 Editor de un Jupyter Notebook.....	57
Imagen 3.39 Colaboratory Notebook con una celda que imprime ‘Hello World!’ y su respectiva respuesta de salida.....	58
Imagen 3.40 Vista general de los servicios y características disponibles en Microsoft Azure	59
Imagen 3.41 Página de inicio de Azure Portal	60
Imagen 3.42 Página de inicio de Azure Marketplace	61

Imagen 3.43 Pantalla de inicio de Microsoft Cognitive Services	62
Imagen 3.44 Página de inicio de Custom Vision.....	63
Imagen 3.45 Ejemplo de un repositorio en GitHub. Tomado de TensorFlow	64
Imagen 3.46 Gráfico completo de commits. Tomado de Loeliger & McCullough (2012).....	65
Imagen 4.1 Ventana de inicio de Visual Studio Code	66
Imagen 4.2 Página principal de Google Colaboratory.....	67
Imagen 4.3 Notebook "Te damos la bienvenida a Colaboratory"	68
Imagen 4.4 Elementos de la pestaña <i>Archivo</i>	68
Imagen 4.5 Elementos de la pestaña <i>Entorno de ejecución</i>	69
Imagen 4.6 Elementos de la pestaña <i>Conectar</i>	69
Imagen 4.7 Página principal de Kaggle.....	70
Imagen 4.8 Opciones del menú +.....	70
Imagen 4.9 Un Notebook recién creado en Kaggle.....	71
Imagen 4.10 Menú <i>Save Version</i>	71
Imagen 4.11 Página del curso de Python Core en SoloLearn.....	72
Imagen 4.12 Página del Curso Profesional de Python en CódigoFacilito	73
Imagen 4.13 Ejemplos de declaración de variables.....	73
Imagen 4.14 Ejemplos de declaración de constantes.....	74
Imagen 4.15 Ejemplos de comentarios.....	74
Imagen 4.16 Ejemplos de listas de datos.....	76
Imagen 4.17 Ejemplos de tuplas de datos	77
Imagen 4.18 Ejemplos de diccionarios de datos.....	77
Imagen 4.19 Ejemplos de conjuntos de datos.....	78
Imagen 4.20 Ejemplo de una condición <i>if / else</i>	78
Imagen 4.21 Ejemplo de ciclo <i>for</i>	79
Imagen 4.22 Ejemplo de ciclo <i>while</i>	79
Imagen 4.23 Importación y versión de NumPy	80
Imagen 4.24 Ejemplo de una matriz multidimensional NumPy	80
Imagen 4.25 Importación y versión de Pandas.....	81
Imagen 4.26 Ejemplo de una <i>Serie</i>	82
Imagen 4.27 Ejemplo de un <i>DataFrame</i> con las fechas como <i>Index</i> en el argumento	82
Imagen 4.28 Ejemplo del uso la función <i>iloc</i>	82
Imagen 4.29 Importación del módulo <i>pyplot</i> de <i>matplotlib</i>	83
Imagen 4.30 Ejemplo de un gráfico sencillo de líneas	83
Imagen 4.31 Subramas que utilizan la API orientada a objetos	84
Imagen 4.32 MIT Stata Center como Udnie, por Francis Picabia	85
Imagen 4.33 Ejemplo con salida 0 tras detectar palabras negativas	86
Imagen 4.34 Ejemplo con salida 1 tras detectar una palabra positiva	87
Imagen 4.35 Autoencoder del conjunto de datos MNIST	88
Imagen 4.36 Un ejemplo de una estructura RNN en donde un codificador representa la pregunta: "how are you?" y un decodificador genera la respuesta: "I am good"	89
Imagen 4.37 Ejemplos de traslación imagen a imagen realizada por formulaciones CycleGAN y Pix2Pix	89
Imagen 4.38 Imágenes de baja resolución generadas por GANs.....	90
Imagen 4.39 Conjunto de datos <i>A Large Scale Fish Dataset</i> en Kaggle	92
Imagen 4.40 Página de inicio de Git	92
Imagen 4.41 Ventana de la ubicación de destino	93
Imagen 4.42 Ventana de selección de componentes	93
Imagen 4.43 Ventana de ajuste del nombre de la rama inicial en un repositorio nuevo	94
Imagen 4.44 Ventana de ajuste del entorno PATH	95
Imagen 4.45 Segunda ventana; ventana sobre la librería para conexiones seguras de HTTPS	95
Imagen 4.46 Ventana sobre el tratamiento de los finales de línea.....	96
Imagen 4.47 Ventana de confirmación de la finalización de la instalación de Git	97

Imagen 4.48 Consola de Git Bash	97
Imagen 4.49 Creación de un directorio a través de Git Bash.....	98
Imagen 4.50 Identificación del usuario con nombre y correo electrónico.....	99
Imagen 4.51 Creación del repositorio local.....	100
Imagen 4.52 Procedimiento para realizar un commit	101
Imagen 4.53 Menú + de GitHub.....	101
Imagen 4.54 Página para la creación de un nuevo repositorio en GitHub.....	102
Imagen 4.55 Página del repositorio <i>seafood-classifiers</i>	102
Imagen 4.56 Conexión entre repositorio local/remoto	103
Imagen 4.57 Petición de inicio de sesión	104
Imagen 4.58 Petición de autorización del GCM.....	104
Imagen 4.59 Confirmación del <i>push</i> del primer commit	105
Imagen 4.60 Página del repositorio <i>seafood-classifiers</i> con el primer commit	105
Imagen 4.61 Clonación del Repositorio de GitHub en Colab	107
Imagen 4.62 Importación de Librerías en Colab	107
Imagen 4.63 Obtención de las rutas de archivos y etiquetas con Colab	108
Imagen 4.64 Primeros diez ejemplos de la obtención de rutas de archivos	109
Imagen 4.65 Primeros diez ejemplos de la obtención de etiquetas.....	109
Imagen 4.66 Explicación del surgimiento de la generalización de la función lambda	109
Imagen 4.67 Generación de Series con Pandas con Colab.....	110
Imagen 4.68 Series de las rutas y las etiquetas.....	110
Imagen 4.69 Generación del DataFrame con Pandas	111
Imagen 4.70 Función para soltar las imágenes GT y su comprobación	112
Imagen 4.71 Métodos de conteo	112
Imagen 4.72 Código para mostrar un elemento de cada clase	113
Imagen 4.73 Gráfico mostrando cada uno de los mariscos distintos del dataset	113
Imagen 4.74 Código para visualizar la cantidad de datos en un gráfico con seaborn.....	114
Imagen 4.75 Gráfico de seaborn con el total de elementos de cada una de las clases	114
Imagen 4.76 División de los datos en tres sets	115
Imagen 4.77 Distribución de los datos entre los sets.....	116
Imagen 4.78 Reescalado de las imágenes.....	117
Imagen 4.79 Código empleado para el módulo <i>flow_from_dataframe</i>	118
Imagen 4.80 Arquitectura de la CNN para la clasificación de mariscos	120
Imagen 4.81 Resumen de la CNN para clasificación de mariscos.....	121
Imagen 4.82 Compilado del Modelo con optimizador Adam	124
Imagen 4.83 Entrenamiento del Modelo	125
Imagen 4.84 Botón <i>Add data</i> del menú	126
Imagen 4.85 Cuadro de diálogo para agregar un dataset al Notebook	126
Imagen 4.86 Importación de Librerías en Kaggle	126
Imagen 4.87 Obtención de las rutas de archivos con Kaggle	127
Imagen 4.88 Primeros cinco ejemplos de la obtención de rutas de archivos en Kaggle.....	127
Imagen 4.89 Generación de Series con Pandas con Kaggle	127
Imagen 4.90 Sección <i>Proyectos</i> de Custom Vision Service.....	128
Imagen 4.91 Cuadro de diálogo <i>Create new project</i>	128
Imagen 4.92 Cuadro de diálogo <i>Create New Resource</i>	129
Imagen 4.93 Cuadro de diálogo <i>Create New Resource Group</i>	129
Imagen 4.94 Cuadro de diálogo <i>Create New Resource</i> con los datos completos	130
Imagen 4.95 Cuadro de diálogo <i>Create new project</i> con los datos completos	131
Imagen 4.95 Cuadro de diálogo <i>Create new project</i> con los datos completos	131
Imagen 4.96 Cuadro de diálogo para añadir etiquetas.....	132
Imagen 4.97 Lista de etiquetas, sin imagen alguna	132
Imagen 4.98 Cuadro de diálogo <i>Image upload</i>	133

Imagen 4.99 Cuadro de diálogo de carga exitosa de las imágenes	133
Imagen 4.100 Lista de etiquetas con datos	134
Imagen 4.101 Cuadro de diálogo de opciones de entrenamiento	135
Imagen 5.1 Resultado de las primeras 5 épocas de entrenamiento del Modelo CNN	136
Imagen 5.2 Resultado de las últimas 5 épocas de entrenamiento del Modelo CNN.....	136
Imagen 5.3 Evaluación del Modelo.....	137
Imagen 5.4 Código para trazar Gráficos de Pérdida y Precisión	138
Imagen 5.5 Gráfica de Pérdida del Modelo CNN	139
Imagen 5.6 Gráfica de Precisión del Modelo CNN.....	139
Imagen 5.7 Código del Reporte de Clasificación y la Matriz de Confusión.....	140
Imagen 5.8 Reporte de Clasificación del Modelo CNN	141
Imagen 5.9 Matriz de Confusión del Modelo CNN	142
Imagen 5.10 Línea de Código para guardar el Modelo CNN	142
Imagen 5.11 Estadísticas de entrenamiento de la Iteración 1	143
Imagen 5.12 Rendimiento de cada Etiqueta.....	144
Imagen 5.13 Parte superior de la página, incluido <i>Quick Test</i>	144
Imagen 5.14 Pez Betta (Morriquitín) sin relación alguna con el dataset, en el cuadro de diálogo de <i>Prueba Rápida</i>	145
Imagen 5.15 Selección de Cloud Shell.....	146
Imagen 5.16 Selección de Crear almacenamiento	146
Imagen 5.17 Selección <i>Abrir editor</i> en la CLI	146
Imagen 5.18 Código escrito en el editor de la CLI.....	147
Imagen 5.19 Código en el Bash de Cloud Shell	147
Imagen 5.20 Marisco a analizar a través de Cloud Shell.....	147
Imagen 5.21 Probabilidades por categoría para la Imagen.....	148
Imagen 5.22 Guardar una copia del Notebook en GitHub	149
Imagen 5.23 Guardado del código del proyecto en GitHub	150
Imagen 5.24 Guardado del código del proyecto en GitHub	150
Imagen 6.25 Confirmación del <i>pull</i> al repositorio local	151

INTRODUCCIÓN

A causa de la emergencia sanitaria global a causa de la pandemia por COVID-19, se otorgó la posibilidad de realizar un proyecto en la Universidad Tecnológica de San Juan del Río en el área de investigación. El proyecto consiste en realizar un clasificador de mariscos haciendo uso de algoritmos y técnicas de Inteligencia Artificial, además de herramientas de desarrollo de software; actualmente usados en la investigación y en la industria.

Este reporte está dividido en cinco capítulos, y a su vez, estos en bastantes subcapítulos con la intención de este posea una estructura detallada de cada aspecto del proyecto.

El primer capítulo se trata sobre la Universidad en donde se llevó a cabo el desarrollo del proyecto, pues se habla de su historia, su misión y su visión, así como de los valores que la representan, su modelo educativo y algunas distinciones que ha obtenido.

El segundo capítulo abarca sobre las generalidades del proyecto y en este capítulo se muestra su nombre y problema a resolver, así como lo que se busca obtener en este proyecto, como se va a resolver y hasta donde se va a llegar.

El tercer capítulo trata sobre todo el marco teórico del proyecto y es aquí donde se brinda la información introductoria necesaria para poder entender terminología y conceptos de Inteligencia Artificial. También se da una breve introducción de las herramientas de software que se utilizan a lo largo del proyecto.

El cuarto capítulo comprende del desarrollo del proyecto y en este capítulo se explica todo lo que se hizo a lo largo de su periodo de duración.

El quinto y último capítulo se muestran las pruebas y los resultados obtenidos de todo lo que se hizo en el capítulo anterior, además de que brinda un cierre al proyecto.

Por último, los archivos de este proyecto están disponibles en <https://github.com/IsaacIsaias/seafood-classifiers>.

CAPÍTULO 1

ANTECEDENTES DE LA UNIVERSIDAD TECNOLÓGICA DE SAN JUAN DEL RÍO

1.1 Historia

La Universidad Tecnológica de San Juan del Río (UTSJR), es una Institución de Educación Superior del municipio de San Juan del Río en el Estado de Santiago de Querétaro, fue fundada el 28 de agosto de 1998 e inició actividades académicas tres días después. Esta Universidad ofrece a los jóvenes egresados del bachillerato, carreras universitarias estrechamente vinculadas con el sector productivo para que en un corto plazo se incorporen al trabajo profesional de la región. Actualmente, se imparten ocho ingenierías y una licenciatura y de las que se distribuyen doce distintas salidas laterales de TSU. Desde el 2015, la UTSJR es dirigida por la rectora M.A.P. Bibiana Rodríguez Montes.

1.2 Misión

Formar personas en educación superior con competencias profesionales, promover la investigación, así como ofrecer servicios tecnológicos, a través de una estrecha vinculación con los distintos sectores del entorno, adaptándose a los cambios y contribuyendo al desarrollo científico y tecnológico.

1.3 Visión

Ser una universidad reconocida nacional e internacionalmente por la capacidad de adaptación, flexibilidad y aprendizaje continuo de sus egresados, que trascienda en la generación y aplicación del conocimiento, atendiendo las necesidades de los sectores público, privado y social.

1.4 Valores

“Actitud Lobo” es una campaña de la Universidad que promueve la integración de la comunidad universitaria con la colonia en la que se encuentra, Vista Hermosa, con el objetivo de que se difundan, preserven y vivan los siguientes valores institucionales:

- Respeto
- Responsabilidad
- Integridad
- Compromiso
- Solidaridad
- Cuidado del Medio Ambiente

1.5 Modelo Educativo

El modelo educativo de la Universidad Tecnológica de San Juan del Río, así como de las demás Universidades Tecnológicas, consta de los siguientes seis aspectos:

1.5.1 Intensidad

Es uno de los atributos que permite desarrollar los contenidos programáticos en 6 cuatrimestres, cumpliendo un promedio de 3,000 horas efectivas de formación.

1.5.2 Continuidad

Le da al egresado la posibilidad de continuar sus estudios a nivel licenciatura en programas educativos afines a su área de formación.

1.5.3 Calidad

No expresada solamente en los rendimientos escolares, debe manifestarse en la responsabilidad profesional, es decir, que el egresado debe cumplir eficazmente sus funciones, aspecto cuya realidad sólo es posible si el alumno adquiere un conocimiento sólido, una práctica eficiente, así como actitudes y valores que le permitan desempeñar un trabajo útil a la sociedad.

1.5.4 Pertinencia

Se logra por la Vinculación con el sector productivo de bienes y servicios, de tal manera que los egresados satisfagan las necesidades reales del entorno de cada universidad. Donde las empresas participan en la definición de los perfiles profesionales, y los planes y programas de estudio.

1.5.5 Polivalencia

Otorga una formación profesional en uno o varios grupos de actividades de los procesos productivos o en actividades generales aplicables a diversas ramas de la producción.

1.5.6 Flexibilidad

Permite la adaptación permanente a los cambios científico-tecnológicos y a los requerimientos profesionales; esto contribuye a incorporar los descubrimientos científicos, las innovaciones tecnológicas y los cambios que se generen en los procesos productivos.

1.6 Certificados y Premios

A lo largo de los años, la Universidad ha obtenido varias certificaciones y premios. Algunas de sus certificaciones y premios son los siguientes:

- Lic. en Innovación de Negocio y Mercadotecnia ante el CACECA
- Ing. de Mecatrónica ante el CACEI
- ISO 9001:2015, Sistemas de Gestión de la Calidad
- 2007, Premio Nacional de Exportación
- 2008, Premio Nacional de la Calidad
- Reconocimiento como Entidad Promotora de Responsabilidad Social (8 años consecutivos)

1.7 Área de Desarrollo

El área de desarrollo del proyecto de estadía es la Inteligencia Artificial haciendo uso del Aprendizaje Profundo, la Visión Computacional y el Procesamiento de Imágenes con el objetivo de aplicar estas tecnologías al sector productivo con el fin de desarrollar proyectos de automatización o al sector de investigación para seguir mejorándolas y encontrar nuevas y más eficientes formas de usarlas.

CAPÍTULO 2

GENERALIDADES DEL PROYECTO

2.1 Nombre del Proyecto

Clasificador de Mariscos por medio de algoritmos basados en Inteligencia Artificial.

2.2 Problema

Desde hace unas décadas, se ha estado tratando de emular la visión humana para aplicarla a proyectos de Inteligencia Artificial con la finalidad de utilizarla para ayudar a automatizar y mejorar procesos donde se involucre la clasificación y segmentación de imágenes, así como la localización y detección de objetos; siendo la Visión Computacional la disciplina principal y el Procesamiento de Imágenes la disciplina paralela, las que se encargan de ello. La Inteligencia Artificial se utiliza en esta área porque logra una gran precisión en los resultados, ya que una de sus aplicaciones es la detección de cáncer en MRIs (imágenes de resonancia magnética) y esto lo hace con la misma precisión que radiólogos altamente capacitados.

2.3 Objetivo General

Clasificar los mariscos de una base de datos proporcionada por Kaggle mediante algoritmos basados en Inteligencia Artificial para discernir los distintos mariscos en las imágenes.

2.4 Metodología

En primer lugar, se debe tener los conocimientos de Python y sus librerías especializadas en Inteligencia Artificial; y Aprendizaje Profundo mínimos necesarios para el desarrollo del código en un Jupyter Notebook, logrando eso a través de diversos cursos y libros. De igual manera, con la documentación y cursos de Git y GitHub, se creará un repositorio en Git en un computador personal para que posteriormente este repositorio se clone y esté en un perfil de GitHub de tal forma que cualquier persona pueda tener acceso a los archivos del proyecto y tenga la posibilidad de clonar el repositorio. Después, se realizarán dos clasificadores, el primer clasificador desarrollado en las plataformas de Google Colab y Kaggle, en el que se utilizará en Colab el repositorio de GitHub, mientras que con Kaggle se utilizará directamente

de ahí el conjunto de datos; y el segundo clasificador hecho en la plataforma de Microsoft Custom Vision Service. Finalmente se realizará una comparativa entre los dos distintos clasificadores previamente realizados.

2.5 Alcance

Desarrollar un código de fácil implementación con Python sobre un modelo de Aprendizaje Profundo (DL) usando redes neuronales convolucionales (CNN), cuya arquitectura se basa en la API Funcional de Keras, API cuyo backend es TensorFlow. El modelo a realizar es un clasificador multiclase de mariscos, cuyo conjunto de datos es obtenido de la plataforma Kaggle. Este modelo de CNNs se espera que se obtenga una precisión superior al 90% y que este sea diseñado y desarrollado en Notebooks basados en Jupyter que se encuentran en plataformas online, para el aprovechamiento de las GPUs gratuitas para el entrenamiento de este.

Desarrollar un segundo clasificador multiclase No-Code con el mismo conjunto de datos con la API de Azure Custom Vision, a través de la plataforma Microsoft Custom Vision Services. Esto con el fin de usar herramientas de la Industria 4.0 en la que no sea necesario implementar código para su desarrollo.

Crear un repositorio local y un repositorio remoto en GitHub para tener un sistema de control de versiones del proyecto y se puedan almacenar todos los archivos para su elaboración.

CAPÍTULO 3

MARCO TEÓRICO

3.1 Inteligencia Artificial (IA)

La Inteligencia Artificial, también conocida como IA, es una tecnología muy poderosa en la actualidad que tuvo sus primeros indicios cuando Alan Turing, padre de la informática teórica y considerado también como el primer padre de la IA; en 1950 publica su artículo “Computing Machinery and Intelligence” en el que propuso la pregunta filosófica, ¿pueden pensar las máquinas? (IBM, 2020)

A causa de esto surgió el experimento de la prueba de Turing, una prueba de habilidad en el que un computador la pasa, si una persona después de formularle algunas preguntas escritas, esta no pueda saber si las respuestas escritas provienen de una persona o de un computador. (Turing, 1950)

Actualmente, seis disciplinas componen la mayor parte de la IA y estas a la vez son capacidades que debe superar una computadora durante una prueba de Turing Total para que se tenga además una interacción con objetos y personas en el mundo real. Las seis capacidades son: el procesamiento de lenguaje natural, la representación del conocimiento, el razonamiento automatizado, el aprendizaje automático, la visión computacional y el reconocimiento de voz; y la robótica. (Russell & Norvig, 2021)

(McCarthy, Minsky, Rochester, & Shannon, 1956), realizaron la primera conferencia científica sobre Inteligencia Artificial en Hanover, New Hampshire; llamando de esa forma a ese nuevo campo de estudio de la ciencia y la ingeniería. Por tal motivo son considerados padres de la Inteligencia Artificial.

Tres años más tarde, Minsky y McCarthy fundaron en el Instituto Tecnológico de Massachusetts (MIT) el Laboratorio de Inteligencia Artificial (AI Lab), lugar donde se hicieron bastantes desarrollos tecnológicos. Hoy en día, se encuentra fusionado con el

Laboratorio de Ciencias de la Computación (LCS), siendo ahora el Laboratorio de Ciencias de la Computación e Inteligencia Artificial del MIT (CSAIL). (MIT CSAIL, 2021)

Finalmente, el mismo McCarthy (2007, pág. 2) dice que la Inteligencia Artificial “es la ciencia y la ingeniería para fabricar máquinas inteligentes, especialmente programas informáticos inteligentes. Está relacionado con la tarea similar de usar computadoras para comprender la inteligencia humana, pero la IA no tiene que limitarse a métodos biológicamente observables”.

3.1.1 Machine Learning (ML)

El Machine Learning, también conocido como el Aprendizaje Automático, es un subconjunto de la Inteligencia Artificial en el que las computadoras aprenden de los datos, generalmente para mejorar su desempeño en alguna tarea estrictamente definida, sin ser programados explícitamente. (Patel, 2019)

3.1.2 Deep Learning (DL)

El Deep Learning, o Aprendizaje Profundo, es un subconjunto del Machine Learning en el que se utilizan exclusivamente múltiples capas de neuronas para extraer patrones y características de los datos en bruto (datos sin procesar) y estas múltiples capas de neuronas estando interconectadas crean redes neuronales artificiales. (Yalçın, 2020)

3.2 Deep Learning

3.2.1 Redes Neuronales Artificiales (ANNs)

Las redes neuronales artificiales constituyen el elemento fundamental del Deep Learning y es un algoritmo especial inspirado en el cerebro humano, por tanto, fue diseñado para simular su mecanismo de trabajo. Bajo este concepto, la neurona es la unidad básica de la estructura de las ANNs y, a su vez, una ANN crea una arquitectura en capas encadenando neuronas en varias capas, para que después una señal pase a través de estas capas y se procese de diferentes formas en cada una de las capas hasta que se genera la salida final requerida.

Las capas ocultas por las ANNs actúan como capas de abstracción, permitiendo así el Deep Learning, el cual es muy demandado actualmente para el desarrollo de herramientas poderosas relacionadas con imágenes, como es el caso de la búsqueda de imágenes mediante el motor de búsqueda de Google o la aplicación Photos de la misma Google. (Ahmad, 2020)

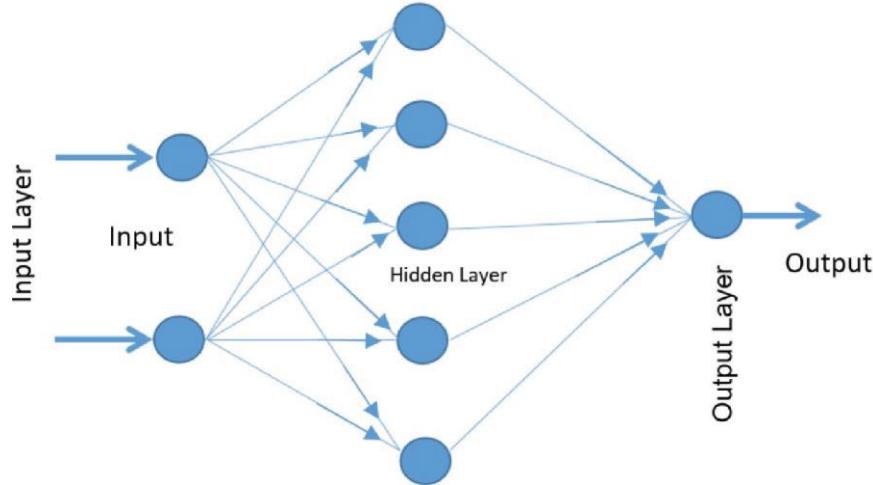


Imagen 3.1 Estructura de una ANN con varias capas. Tomado de Kumar, Upadhyay, & Kumar (2020)

3.2.2 Redes Neuronales Convolucionales (CNNs)

Las ANNs son la base de las redes neuronales convolucionales, CNNs, pero con algunos cambios para que estas sean adecuadas para analizar bastas cantidades de datos, especialmente con los que poseen algunas propiedades espaciales, como lo son las imágenes.

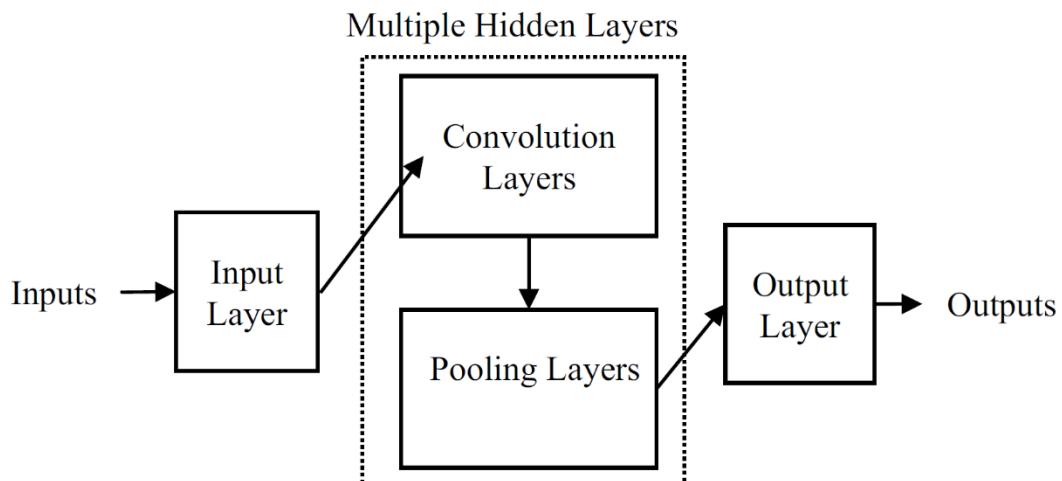


Imagen 3.2 Estructura básica de una CNN. Tomado de Borra, Thanki, & Dey (2019)

Las CNNs se componen principalmente de capas convolucionales y se encargan de filtrar sus entradas de capa con la finalidad de encontrar características útiles dentro de estas entradas. A este procedimiento se le llama convolución. (Zafar, Tzaniou, Burton, Patel, & Araujo, 2018)

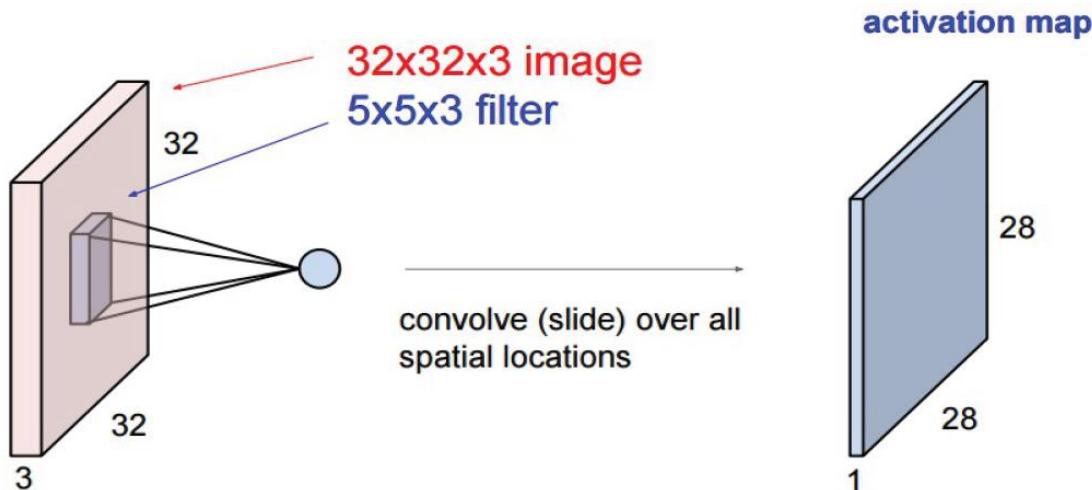


Imagen 3.3 Diagrama de una operación convolucional 2D en una imagen y su resultado

La manera en que nosotros vemos imágenes, o identificamos objetos en ellas, es muy diferente a como lo realiza un computador, pues este ve a las imágenes como una matriz de píxeles. En el caso del diagrama de la Imagen 3.3, se tiene una imagen RGB (Red, Green, Blue) con dimensiones de 32×32 , siendo así una matriz tridimensional de $32 \times 32 \times 3$, de donde se tienen 32 filas, 32 columnas y 3 son del valor del canal RGB. En el rango de píxeles de la imagen, el valor depende de los tipos de datos de la imagen, proporcionando bits de la imagen. Para 8 bits, las imágenes de datos tienen valores que están entre el 0 y el 255 para cada uno de estos números. Estos valores describen como elemento vectorial la intensidad del píxel en cada fila y columna y para clasificar estos píxeles, se genera una propiedad única a partir de un algoritmo de clasificación.

En el caso de las CNNs, las capas convolucionales construyen conceptos más abstractos. Los pasos seguidos en las CNN en un dato de entrada dado, son una serie de operaciones convolucionales, operaciones no lineales, operaciones de capas de agrupación y, por último, se aplican los pasos de capa completamente conectada para obtener la salida.

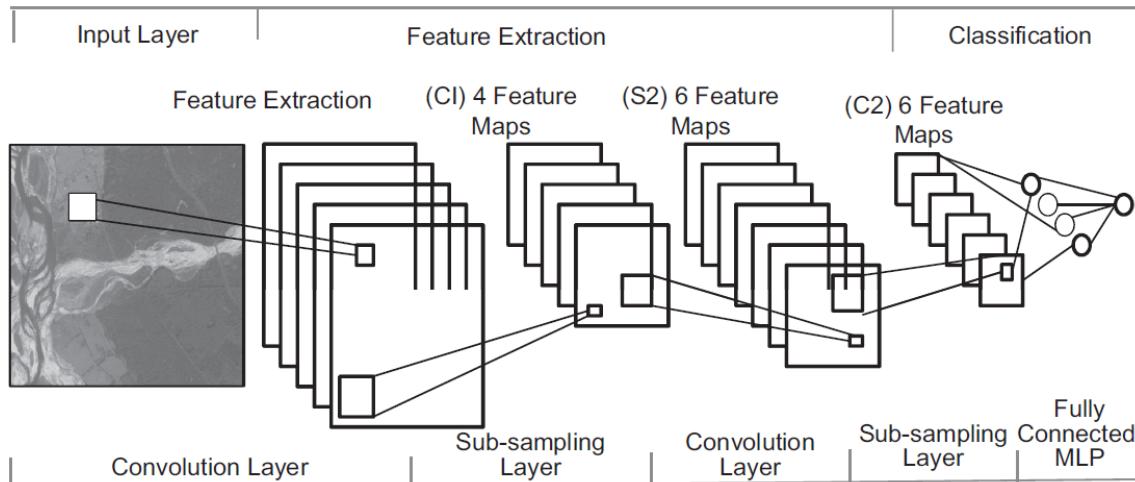


Imagen 3.4 Arquitectura de una CNN multicapa. Tomado de Kumar, Upadhyay, & Kumar (2020)

La operación de la capa convolucional es siempre la primera en la que se introduce una imagen. La lectura de una imagen se realiza desde la esquina superior izquierda, ya que esto toma menos tiempo leerla. El siguiente paso es seleccionar una matriz pequeña llamada kernel. El kernel, también conocido como filtro, genera una salida de convoluciones, mientras se mueve a lo largo de la imagen de entrada. Su objetivo es el de multiplicar sus valores con los valores de píxeles originales de la imagen en esa región. Se calcula la suma ponderada, multiplicando los coeficientes del kernel con los valores de la imagen y, finalmente, se genera un único número a partir de esta operación.

El kernel tiene dos parámetros, llamados zancada y tamaño, siendo ‘stride’ y ‘size’ sus respectivos nombres en inglés. Una zancada es el número de píxeles que se va a mover en cada iteración, es decir, son los movimientos de unidad que va a realizar el filtro y pueden ser de cualquier valor, por ejemplo, una zancada de una unidad produce una imagen de casi el mismo tamaño, mientras que una zancada de longitud dos produce la mitad del tamaño, por lo que, si se quiere tener el mismo tamaño de entrada, es necesario que se rellene la imagen para que ayude en esa parte. Por último, se tiene el tamaño del kernel y este puede ser de cualquier dimensión y en forma de un rectángulo.

En primera instancia, el kernel ha utilizado valores de píxeles de la esquina superior izquierda únicamente, pero tiene que seguir moviéndose a lo largo de la imagen, por lo que tiene que

hacer zancadas para poder realizar dichas operaciones. Posteriormente, tras ser aplicado el kernel en toda la imagen, se obtiene una salida en forma de matriz, matriz que por lo general es más pequeña que la matriz de entrada, pues se tiene menor altura y menor anchura, aunque más canales. Todo este proceso logra que se identifiquen límites y colores simples de la imagen de entrada con respecto a la percepción humana. (Kumar, Upadhyay, & Kumar, 2020)

Los parámetros aprendidos en una capa convolucional son los pesos del kernel de una capa. Durante el entrenamiento de la CNN, los valores de estos kernels se ajustan automáticamente para extraer la información más útil para la tarea en cuestión. Ya si se desea obtener propiedades muy específicas de las características de la imagen, se requiere un nivel más alto de red CNN.

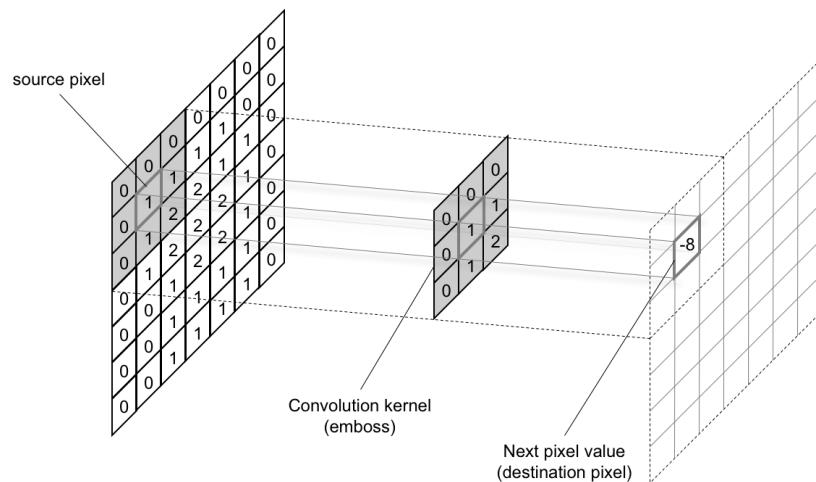


Imagen 3.5 Ejemplo de operación de filtrado. Tomado de Shanmugamani (2018)

En el siguiente par de imágenes se puede observar un ejemplo de la aplicación de un kernel. La primera imagen será la imagen muestra donde se le aplicará este y la segunda imagen es el resultado tras su aplicación. El kernel, o filtro, que se le aplicará a la primera imagen es de la forma 3x3 y es el siguiente:

$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$



Imagen 3.6 Imagen muestra a la que se le va a aplicar el kernel.

Tomado de “Redes Neuronales Convolucionales” por IBM Development, 2018, <https://developer.ibm.com/es/articles/convolutional-neural-network-vision-recognition/>



Imagen 3.7 Imagen del resultado de la aplicación de un solo pase del kernel. Tomado de IBM Development (2018)

En las redes neuronales tradicionales, se tiene que convertir cualquier dato de entrada en un solo vector unidimensional, logrando así que se pierda toda la información espacial importante después de que este vector se envíe a una capa completamente conectada.

Además, cada píxel tendría un parámetro por neurona que conduciría a un abismal número de parámetros en un modelo grande de cualquier tamaño o profundidad.

La red CNN consta de varias operaciones convolucionales, operaciones de agrupación y capas no lineales. En las CNN, la salida de la primera capa se convierte en la entrada para la siguiente capa, aunque la capa siguiente pueda no ser la misma que la primera, y esto sigue sucediendo con las capas consecutivas hasta que se llega a la última capa.

En la activación de la CNN, una función está presente como capa no lineal, después de cada operación convolucional que se le va a aplicar. (Glorot, Bordes, & Bengio, 2011; Krizhevsky, Sutskever, & Hinton, 2017; Nair & Hinton, 2010; LeCun, Bengio, & Hinton, 2015; Ramachandran, Zoph, & Le, 2017) Esta función produce un límite de decisión no lineal a través de combinaciones no lineales del peso y las entradas. Sin el límite de decisión no lineal, la red no podría modelar la variable de respuesta. Después de la capa no lineal, la capa de agrupación viene a continuación en la CNN. (Lin, Chen, & Yan, 2013)

La operación de agrupación reduce el tamaño de la imagen mediante la operación de submuestreo. En el submuestreo, la imagen se comprime de tal forma en la que sus detalles se redujeron como si fueran imágenes menos detalladas. Esta operación de submuestreo se finaliza porque ya se han identificado varias características en la operación de convolución anterior y ya no se necesita una imagen con información detallada.

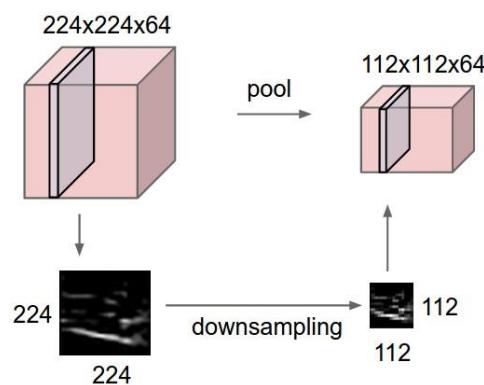


Imagen 3.8 Ejemplo en el que un volumen de entrada es agrupado con un kernel y una zancada de tamaño 2. Tomado de Stanford University (2021)

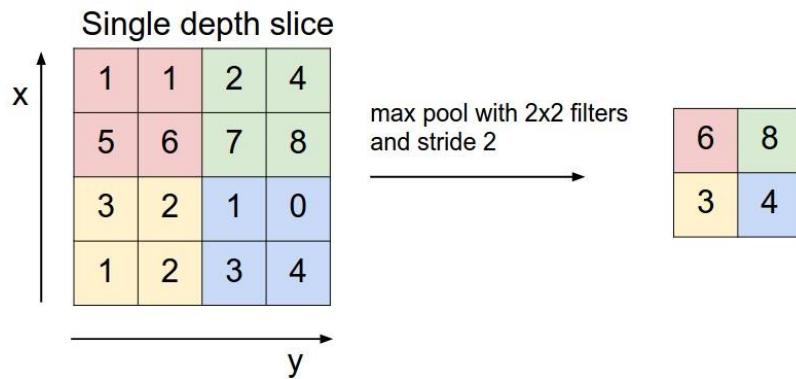


Imagen 3.9 Ejemplo de la operación de agrupación más común, max pooling, con una zancada de 2. Tomado de “CS231n Convolutional Neural Networks for Visual Recognition” por Stanford University, 2021, <https://cs231n.github.io/convolutional-networks/>

En la CNN se aplican operaciones no lineales y capas de agrupación después de varios pasos de operaciones convolucionales. Por consiguiente, se debe aplicar la última capa de operación obligatoria, también conocida como capa completamente conectada. La capa completamente conectada proporciona un vector dimensional de tamaño N, donde N representa el número total de clases a identificar. Con esto último, se encarga de brindar la información de salida final de las redes convolucionales.

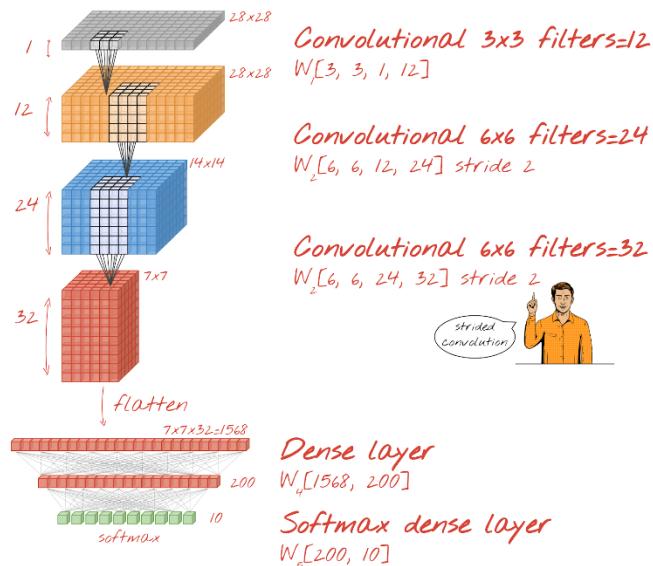


Imagen 3.10 Red convolucional para el reconocimiento de dígitos escritos a mano. Tomado de (Gorner, 2021)

Finalmente, en la Imagen 3.10 se observa un ejemplo de una red convolucional para el reconocimiento de dígitos escritos a mano, en la que se usaron tres capas convolucionales en la parte superior, una capa de lectura softmax en la parte inferior y conectados con una capa completamente conectada.

3.2.3 Arquitecturas ConvNet

Las redes convolucionales como se han visto se componen principalmente de tres tipos diferentes de capas, las capas convolucionales, las capas de agrupación y las capas totalmente conectadas. También se compone de funciones de activación y se le puede considerar una cuarta capa, debido a que estas funciones aplican la no linealidad por elementos.

La forma más común de una arquitectura ConvNet es en la que se apilan algunas capas ConvRelu, siguiéndose de capas de agrupación, y repitiendo así este patrón hasta que la imagen de entrada se haya fusionado espacialmente a un tamaño pequeño. Después, en algún momento en su diseño, es común realizar una transición a capas totalmente conectadas. La última capa totalmente conectada contiene la salida, como el número total de clases. En otras palabras, la arquitectura de ConvNet más común sigue el patrón siguiente:

$$\text{INPUT} \Rightarrow [[\text{CONV} \rightarrow \text{RELU}] * N \rightarrow \text{POOL?}] * M \Rightarrow [\text{FC} \rightarrow \text{RELU}] * K \Rightarrow \text{FC}$$

De donde: *, indica repetición; POOL?, indica una capa de agrupación opcional; $N \geq 0$, y usualmente, $N \leq 3$; $M \geq 0$; $K \geq 0$, y usualmente, $K < 3$.

En este campo de las redes convolucionales, existen varias arquitecturas que poseen ya un nombre, las arquitecturas más comunes son:

- LeNet-5. Esta arquitectura fue introducida con el nombre de LeNet en 1998 por LeCun, Bottou, Bengio, & Haffner, se le considerada de las primeras aplicaciones exitosas de redes convolucionales y su principal uso es en el reconocimiento óptico de caracteres, por ejemplo, para la lectura de códigos postales, dígitos, entre otros usos.
- AlexNet. Esta arquitectura fue el primer trabajo que popularizó las redes convolucionales en el campo de la visión artificial y fue desarrollada por Krizhevsky, Sutskever, & Hinton en el 2012. Esta arquitectura se presentó en el ILSVRC 2010, ImageNet Large Scale Visual Recognition Competition, en donde se clasificaron 1.2

millones de imágenes de alta resolución de un conjunto de entrenamiento de 1000 clases distintas. Dos años más tarde, se presentó una variante de ese mismo modelo el cual superó por mucho al segundo finalista, ya que su tasa de error ‘Top 5’ fue del 16% en comparación con la del segundo lugar con un 26%. La tasa de error ‘Top 5’, o de los cinco primeros, es el porcentaje de imágenes en las que la etiqueta correcta no es una de las cinco etiquetas más probables del modelo. (Russakovsky & Deng, 2015)

- ZF Net. En el ILSVRC 2013 el ganador fue una versión mejorada del AlexNet, teniendo una mejoría en los hiperparámetros, en particular por la expansión del tamaño de las capas convolucionales de en medio y hacer que la zancada y el tamaño del kernel de la primera capa, fueran más pequeños. Fue hecha por Zeiler & Fergus.
- GoogLeNet. El ganador del ILSVRC 2014 fue una red convolucional de Google, hecha por (Szegedy, y otros, 2015). El desarrollo del Módulo Inception fue su principal aporte y este redujo el número de parámetros en la red. Además, se utiliza la Agrupación Promedio, el Average Pooling en inglés, en lugar de las capas completamente conectadas en la parte superior de la ConvNet, ocasionando que sean eliminados una basta cantidad de datos sin importancia. La versión más reciente es la Inception-v4. (Szegedy, Ioffe, Vanhoucke, & Alemi, 2017)
- VGGNet. El segundo lugar del ILSVRC 2014 fue la red realizada por (Simonyan & Zisserman, 2015). La principal contribución de esta red fue mostrar de que la profundidad de la red es importante para que se pueda tener un buen desempeño. Su mejor red final contiene 16 capas CONV → FC y presenta una arquitectura homogénea debido a que solo realiza convoluciones 3x3 y agrupaciones 2x2, desde el principio hasta el final. Este modelo previamente entrenado está disponible en Caffe para su uso Plug and Play, es decir, que se puede utilizar inmediatamente sin la necesidad de que el usuario tenga que configurar algo. Una desventaja de esta arquitectura es que es más costoso de evaluar, y usa más memoria y parámetros.
- ResNet. Residual Network fue desarrollada por (He, Zhang, Ren, & Sun, 2016) y fue la red ganadora del ILSVRC 2015. ResNet cuenta con conexiones de salto especiales y un uso intensivo de Batch Normalization, o Normalización en Lotes. (Ioffe &

Szegedy, 2015) Los ResNets son actualmente la opción predeterminada para usar ConvNets en la práctica, desde el 10 de mayo del 2016.

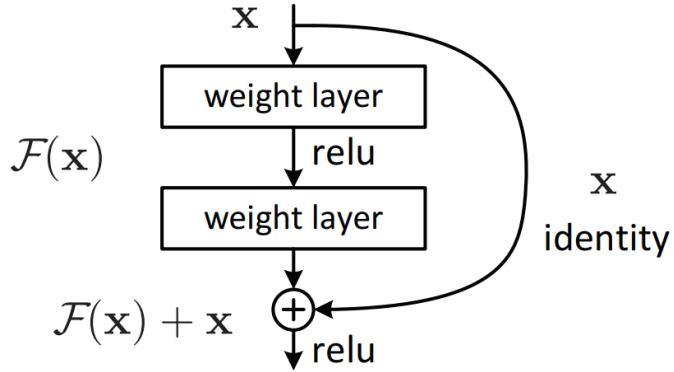


Imagen 3.11 Aprendizaje Residual, un componente. Tomado de
He, Zhang, Ren, & Sun (2016)

3.2.4 Consideraciones Computacionales

Un desafío importante en el diseño de redes neuronales es el tiempo de ejecución requerido para entrenar la red. Para esto se debe considerar que en una gran parte de los modelos la mayor parte del trabajo pesado computacional se carga al frente durante la fase de entrenamiento, y la fase de predicción es a menudo computacionalmente eficiente, porque requiere una pequeña cantidad de operaciones, dependiendo del número de capas. Esto es importante de saber, porque la fase de predicción suele ser mucho más crítica en comparación con la fase de entrenamiento. Un ejemplo de esto es el que es más importante clasificar una imagen en tiempo real, con un modelo prediseñado; aunque la construcción real de este modelo hubiera requerido un conjunto de datos con millones de imágenes en varias semanas. (Aggarwal, 2018)

En años recientes, los avances en la tecnología de hardware, como las GPUs, Unidades de Procesamiento Gráfico, han sido de gran ayuda. Las GPUs son procesadores de hardware especializados que pueden acelerar significativamente los tipos de operaciones que se usan generalmente en las redes neuronales. Estas son programadas utilizando CUDA, la Arquitectura Unificada de Dispositivos de Cómputo, y OpenCL, el Lenguaje de Computación Abierto. Las GPUs ejecutan procesos en paralelo y pueden realizar una gran

cantidad de operaciones a la vez, lo que resulta en un procesamiento más rápido en comparación con las CPUs, Unidades Central de Procesamiento, que realizan principalmente procesamiento en serie. Una diferencia entre estas dos unidades, es que una CPU realiza muchos tipos diferentes de cálculos, mientras que una GPU se especializa en un cálculo determinado, como el procesamiento de imágenes. (Kar, 2020)

Durante la creación de arquitecturas ConvNet, se debe tomar en cuenta el cuello de botella de memoria que se puede generar, para ello hay tres fuentes principales de memoria que se le puede dar seguimiento y tener un control del espacio de la memoria:

- Tamaño de volumen intermedios. Son el número bruto de activaciones en cada capa de la ConvNet, y también sus gradientes. Por lo general, la mayoría de las activaciones se encuentran en las primeras capas de la ConvNet, y estas se mantienen porque son necesarias para la retropropagación, pero una buena implementación en donde se ejecute una ConvNet solo en el momento de la prueba, podría reducir esta retropropagación, almacenando solo las activaciones actuales en cualquier capa y descartando las actividades anteriores en las capas inferiores.
- Tamaño de los parámetros. Son los números que contienen los parámetros de la red, sus gradientes durante la retropropagación y, por lo general, también una caché de pasos si la optimización está usando momentum, Adagrad, o RMSProp.
- Cada implementación de ConvNet tiene que mantener una memoria con diversa cantidad de información, como lo puede ser los lotes de datos de imagen o sus versiones aumentadas, entre otros.

Ya una vez que se obtenga el número total de valores tras haber revisado los tres aspectos anteriores, se debe convertir ese total a una equivalencia en GB para verificar que la GPU que se tiene sea necesaria para la ejecución de la red. Si la red no encaja, una heurística común para hacer que esté dentro del límite de memoria es disminuyendo el tamaño del lote, ya que las activaciones suelen consumir la mayor parte de la memoria. (Stanford University, 2021)

3.3 Procesamiento de Imágenes

(Dey, 2018) define simplemente al procesamiento de imágenes a la manipulación, análisis, interpretación y procesamiento automáticos de imágenes a través de algoritmos y códigos en una computadora. Este posee diferentes aspectos, como el almacenamiento, representación, extracción de información, manipulación, mejora, restauración e interpretación de imágenes.

Este posee muchas aplicaciones en diversas disciplinas y campos de la ciencia y tecnología, como lo son la robótica, la teledetección, la fotografía computacional, la televisión e inspección industrial. También podemos encontrar aplicaciones en los campos médicos/biológicos, por ejemplo, en los diagnósticos médicos, las radiografías y las tomografías computerizadas. Un ejemplo aplicado en la industria es Facebook, en la que sus redes sociales Facebook e Instagram, aplicaciones donde en todo momento son subidas millones de imágenes, se innovan y usan muchos algoritmos de procesamiento de imágenes para procesar esas imágenes subidas. Finalmente, el procesamiento de imágenes también ayuda en métodos de autenticación como lo son la autenticación de huellas dactilares, el reconocimiento facial, entre otros.

3.3.1 Visión Computacional

Dice (Shanmugamani, 2018) que la visión computacional, visión por computadora o visión artificial, es la ciencia de comprender o manipular imágenes y videos. Esta disciplina se encarga de que las máquinas adquieran una capacidad a nivel humano para visualizar, procesar y analizar imágenes y videos.

Los dos problemas centrales de la visión computacional son la reconstrucción y el reconocimiento. En la reconstrucción, un agente construye un modelo del mundo a partir de una imagen o un conjunto de imágenes; y en el reconocimiento, un agente hace distinciones entre los objetos que encuentra basándose de la información visual. (Russell & Norvig, 2021)

El uso del Deep Learning para la visión computacional se puede categorizar entre clasificación, detección, segmentación y generación de objetos, tanto en imágenes como en videos. Sin embargo, uno de los problemas centrales del procesamiento de imágenes, y que cuenta con una amplia variedad de aplicaciones prácticas, es la clasificación de imágenes,

por tanto, todas estas posibles categorías dentro del Deep Learning, aparentemente, pueden reducirse simplemente a una clasificación de objetos, ya que esta está presente en cada una de ellas.

También posee muchas aplicaciones interesantes, de las que se destacan la conducción autónoma, la realidad aumentada y la inspección industrial. (Shanmugamani, 2018)

3.3.2 Clasificación de Imágenes

La clasificación de imágenes se refiere al proceso de asignar una etiqueta a una imagen en función de su contenido visual y esta es una tarea, por lo general, de Aprendizaje Automático supervisado.

En un problema de clasificación de imágenes, un modelo de Deep Learning aprende las clases de las imágenes de una manera gradual utilizando su arquitectura de capa oculta. En primer lugar, este modelo extrae automáticamente características de bajo nivel, como lo pueden ser regiones claras u oscuras, luego extrae características de alto nivel, como los bordes. Posteriormente, este extrae las características de más alto nivel, tales como las formas, para que finalmente se pueda realizar la clasificación.

Cada nodo o neurona artificial en el modelo representa un pequeño aspecto de la imagen completa, cuando se llegan a juntar todas las neuronas son capaces de representar totalmente la imagen. Además, cada nodo de la red tiene un peso asignado, estos pesos representan el peso real de la neurona con respecto a su relación con la salida y se pueden ajustar mientras los modelos se van desarrollando. (Dey, 2018)

Los problemas en la clasificación de imágenes se pueden categorizar en dos grandes grupos. El primero es cuando se requiere hacer una clasificación solo entre dos clases, por ejemplo, el identificar en una imagen si es perro o gato; y se le conoce como clasificación binaria. El segundo es la clasificación multiclasa, por lo que se debe hacer una clasificación entre más de dos clases, un claro ejemplo puede ser el de identificar correctamente un dígito, ya que se dispone de 10 distintos elementos. Cabe mencionar que el tipo de clasificación a escoger podría limitar el tipo de algoritmo de aprendizaje que se podría utilizar y el rendimiento que se desea obtener. (Rivas, 2020)

A continuación, se muestran dos ejemplos distintos de implementaciones de clasificadores.

El primero está basado en las características y en él, un algoritmo de generación de características extrae un conjunto de características que se utilizarán para representar una imagen. Luego un algoritmo de Aprendizaje Automático supervisado clasifica dicha imagen usando las características previamente extraídas.

El segundo es un clasificador de Deep Learning que no tiene preprocesamiento y solamente se entrena la red neuronal de extremo a extremo para realizar la clasificación.

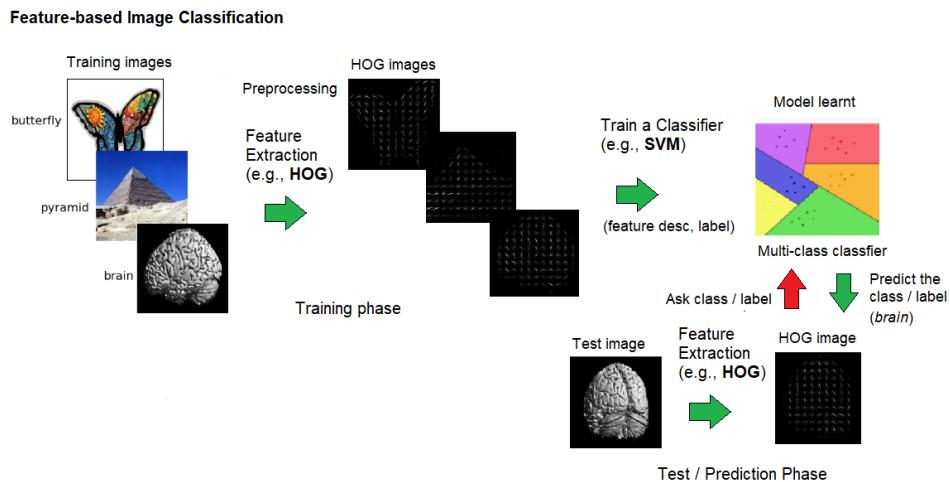


Imagen 3.12 Diagrama de un clasificador de imágenes basado en características.

Tomado de Dey (2020)

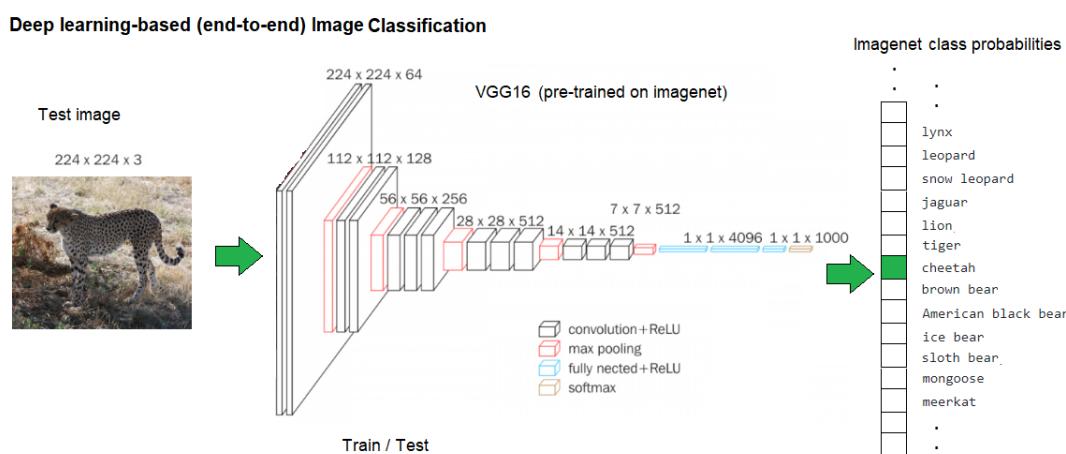


Imagen 3.13 Diagrama de un clasificador de imágenes basado en Deep Learning. Tomado de Dey

(2020)

Así como hay dos tipos principales de problemas en la clasificación, también existen dos tipos de clasificadores, los supervisados y los no supervisados. La principal distinción de estos clasificadores es el uso que se le da a los conjuntos de datos etiquetados, en el supervisado se utilizan estos datos tanto en la entrada como en la salida, mientras que el no supervisado no lo hace.

3.3.2.1 Clasificadores Supervisados

La clasificación supervisada proporciona a un algoritmo un conjunto de muestras de datos o características extraídas de las muestras de datos, junto con sus nombres de etiqueta correspondientes. (Dey, Bhatt, & Ashour, 2018) Este algoritmo analiza las muestras en un espacio de características multidimensional para la extracción de propiedades estadísticas y reglas de decisión, que luego se pueden utilizar para determinar la clase de la nueva muestra de datos. La ‘máxima verosimilitud’ y la ‘mínima distancia’ son métodos comunes para categorizar una imagen completa utilizando los datos de entrenamiento. (Karbhari & Ansari, 2009)

La principal ventaja la clasificación supervisada es que se pueden detectar errores en la clasificación y se pueden corregir dichos errores, mientras que su principal desventaja es la gran cantidad de tiempo que se necesita para realizarlos y que son muy costosos, además introducen errores humanos debido a que, para la selección del conjunto de entrenamiento, previamente una persona puede considerar, o no, todas las distintas condiciones y categorías de clases.

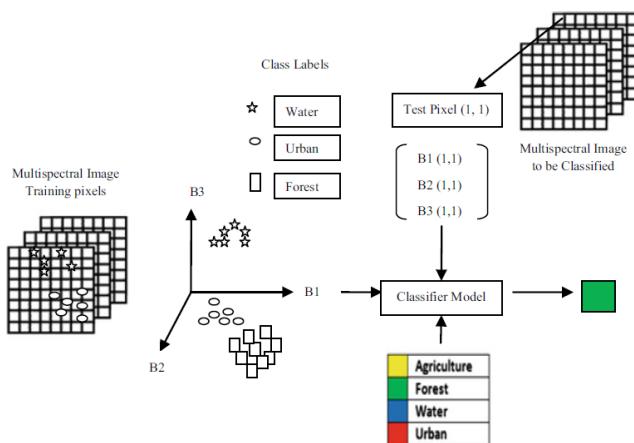


Imagen 3.14 Clasificación Supervisada

3.3.2.2 Clasificadores No Supervisados

La clasificación no supervisada es un proceso totalmente automatizado sin el uso de datos de entrenamiento. Usando un algoritmo adecuado, las características específicas de una imagen se detectan sistemáticamente durante la etapa de procesamiento de imágenes. (Karbhari & Ansari, 2009)

Los algoritmos más frecuentemente usados en clasificadores no supervisados son el K-medias y el ISODATA, un algoritmo usado para reconocimiento de patrones multiespectrales.

Algunas ventajas de estos tipos de clasificador es que principalmente se necesita una mínima cantidad de información inicial para realizarlo, se elimina el error humano debido a que prácticamente todo lo hace una computadora y que es mucho más rápido y sencillo ejecutarlo.

En contraparte, las desventajas son que alguien forzosamente necesita asignar una clase a los clústeres, el cual es una tarea muy tardada y demandante. Otra es que posiblemente los clústeres no coincidan con las clases que nosotros los humanos podemos identificar claramente.

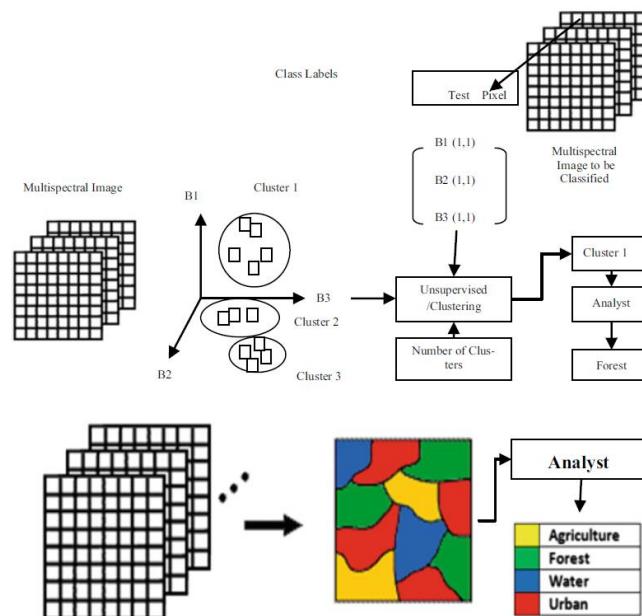


Imagen 3.15 Clasificación No Supervisada

3.3.3 Segmentación de Imágenes

La segmentación de imágenes consiste en dividir una imagen en distintas regiones o categorías, las cuales corresponden a diferentes objetos o partes de objetos. Cada región contiene píxeles con atributos similares, y cada píxel de una imagen se le asigna una de estas categorías.

Se considera una buena segmentación cuando los píxeles de una misma categoría tienen parecidos valores de intensidad y forman una región conectada, mientras que los píxeles vecinos que están en categorías distintas tienen valores diferentes. Esto se hace generalmente para simplificar la representación de la imagen en algo más fácil de analizar. Si hizo adecuadamente, las demás etapas del análisis de imágenes se simplifican, lo que significa que la confiabilidad y la calidad de la segmentación determina si el análisis de una imagen será exitoso. (Dey S. , 2020)

Las técnicas de segmentación pueden ser no contextuales, donde no se consideran las relaciones espaciales entre las características de una imagen y los píxeles de grupo solo con respecto a algunos atributos globales, por ejemplo, el nivel de gris o color; o contextuales, donde se aprovechan las relaciones espaciales, por ejemplo, el agrupamiento de píxeles espacialmente cercanos con niveles de gris similares.

Una segmentación semántica es la tarea donde se predicen etiquetas píxel a píxel. En las siguientes dos imágenes se muestra un ejemplo de esto, siendo la segunda imagen la segmentación semántica de la primera imagen.



Imagen 3.16 Imagen muestra donde se aplicará la segmentación semántica

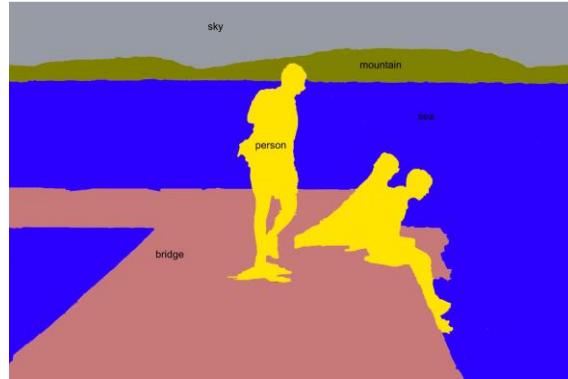


Imagen 3.17 Segmentación semántica de la Imagen
3.16. Tomado de Shanmugamani (2018)

Como se ve en la Imagen 3.17, cada píxel tiene una etiqueta. En este caso las etiquetas son cielo, montaña, mar, puente y persona. La segmentación semántica etiqueta los píxeles de forma independiente, aunque en este caso, las personas no se pueden distinguir una de otra, ya que se encuentran etiquetadas de la misma manera.

Para que las personas se puedan distinguir entre sí, se debe realizar una segmentación de instancia, tarea que consiste en segmentar cada instancia con una etiqueta píxel a píxel. Con esto, la segmentación de instancias puede considerarse una extensión de la detección de objetos con etiquetas a nivel de píxel.

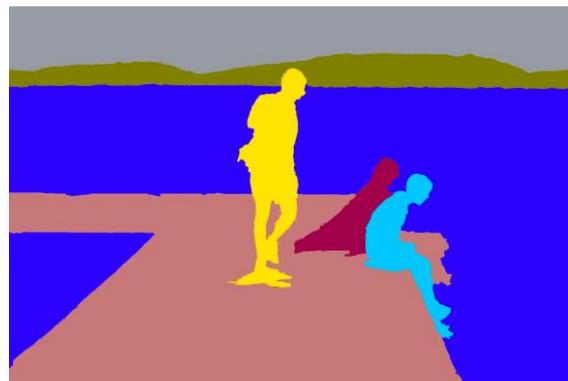


Imagen 3.18 Segmentación de instancias donde cada instancia se distingue de una misma etiqueta
(persona)

Los algoritmos de segmentación generalmente se dividen en dos propiedades básicas de los valores de nivel de gris, las discontinuidades y la similitud.

3.3.3.1 Discontinuidades

Las técnicas basadas en discontinuidades intentan encontrar límites completos que encierran regiones relativamente uniformes asumiendo cambios abruptos de los niveles de gris en cada límite. (Efford, 2000)

Existen tres tipos básicos de discontinuidades: los puntos, las líneas y los bordes. La detección se basa en convolucionar la imagen con un kernel, también conocido como matriz convolucional o máscara espacial.

Un kernel 3x3 general es de la forma:

$$\begin{bmatrix} W_{-1,-1} & W_{-1,0} & W_{-1,1} \\ W_{0,-1} & W_{0,0} & W_{0,1} \\ W_{1,-1} & W_{1,0} & W_{1,1} \end{bmatrix}$$

La respuesta del kernel en cualquier punto (x, y) en la imagen es:

$$R_{x,y} = \sum_{i=-1}^1 \sum_{j=-1}^1 p(x-i, y-j)w(i,j)$$

Un punto ha sido detectado en la ubicación $p(i, j)$ en el que el kernel es centrado, si

$|R| > T$, donde T es un umbral no negativo y R se obtiene con el siguiente filtro:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Para la detección de puntos, la idea principal es que el nivel de gris de un punto aislado sea bastante diferente al nivel de gris de sus vecinos.

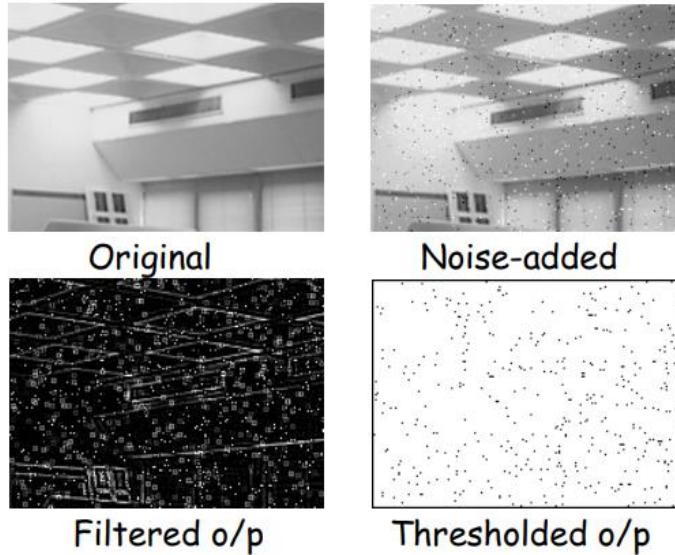


Imagen 3.19 (Sup. Izq.) Imagen original; (Sup. Der.) Ruido agregado; (Inf. Izq.) Salida del filtro; (Inf. Der.) Salida del umbral

Los kernels para detectar distintos tipos de líneas son los siguientes:

- Líneas horizontales:

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

- Líneas de 45° :

$$\begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

- Líneas verticales:

$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

- Líneas de -45° :

$$\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

Si, en un cierto punto de la imagen, $|R_i| > |R_j|$ para todo $j \neq i$, se dice que es más probable que ese punto esté asociado con una línea en la dirección del kernel i.

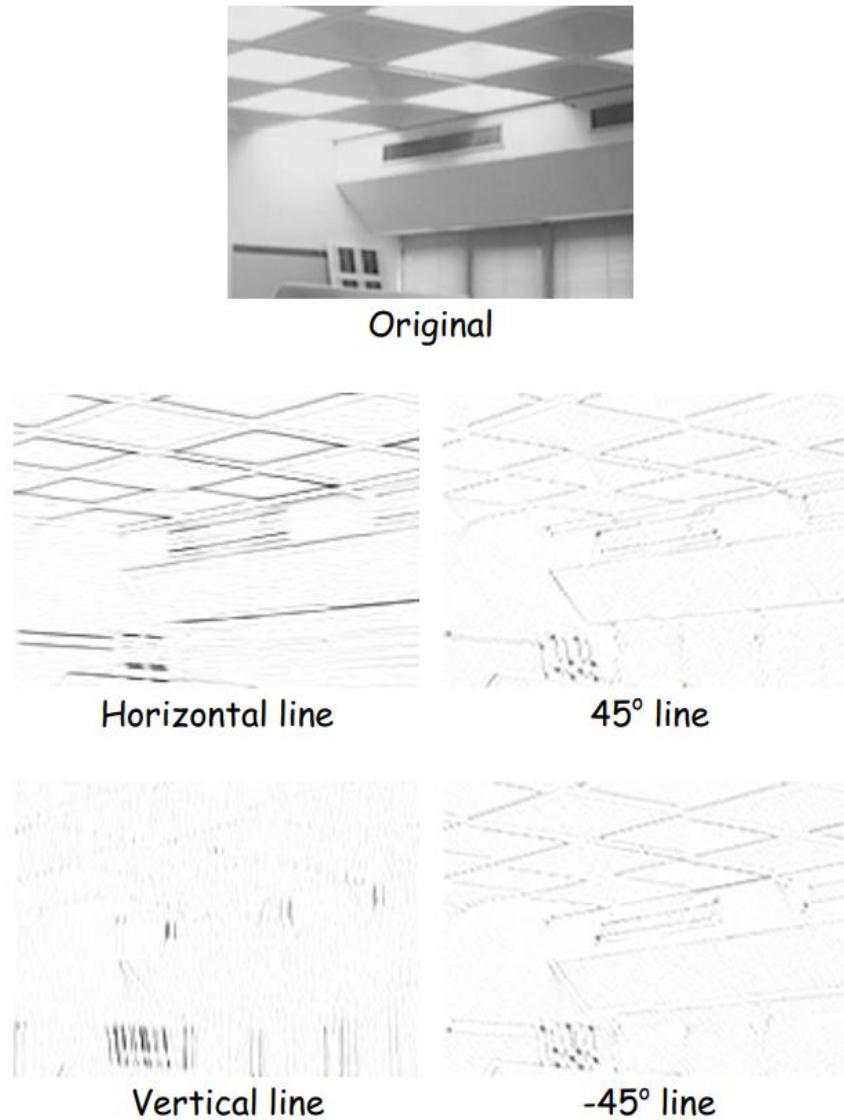


Imagen 3.20 Imagen muestra con cuatro ejemplos de distintas salidas tras la aplicación de un distinto kernel en específico

El tercer y último tipo básico de discontinuidad es la detección de bordes y es el método más común para detectarlos porque los puntos aislados o las líneas de un píxel de ancho son raros.

Esta discontinuidad localiza cambios bruscos en la función de intensidad y los bordes son píxeles donde el brillo cambia abruptamente. Un cambio de la función de la imagen se puede describir mediante una pendiente que apunta en la dirección del mayor crecimiento de la función de la imagen.

Un borde es una propiedad adjunta a un píxel individual y se calcula a partir del comportamiento de la función de imagen en una vecindad del píxel. La magnitud de la primera derivada detecta la presencia del borde y el signo de la segunda derivada determina si el píxel del borde se encuentra en el signo oscuro o claro.

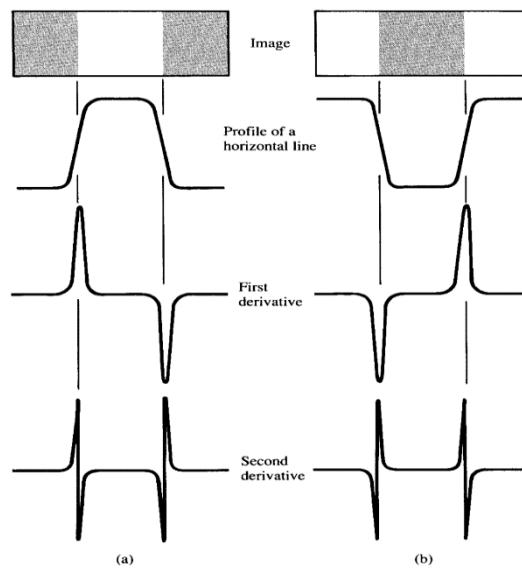


Imagen 3.21 Detección de bordes por operadores de derivada: (a) Franja clara sobre un fondo oscuro; (b) Franja oscura sobre un fondo claro

3.3.3.2 Similitud

Las técnicas basadas en similitudes intentan crear directamente estas regiones uniformes agrupando píxeles conectados que satisfacen ciertos criterios de similitud. (Efford, 2000)

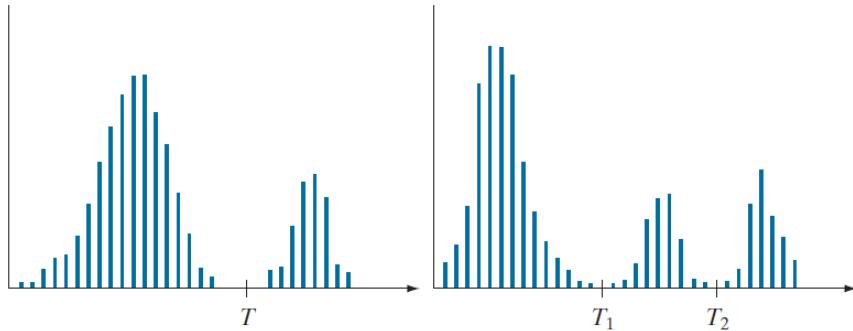
La segmentación de imágenes basada en la similitud se puede realizar de tres formas distintas y estas son el método del valor del umbral (thresholding), el crecimiento de región (region growing), y la división y fusión de región (region splitting and merging).

El thresholding es uno de los enfoques más importantes para la segmentación de imágenes. Si los píxeles de fondo y de objetos tienen niveles de gris agrupados en dos modos dominantes, estos pueden ser separados fácilmente con threshold (umbral).

El thresholding se puede ver como una operación que involucra pruebas contra una función T de la forma:

$$T = T[x, y, p(x, y), f(x, y)]$$

Donde, $f(x, y)$ es el nivel de gris del punto (x, y) ; y $p(x, y)$ denota alguna propiedad local de este punto, como el promedio del nivel de gris de un centro vecinal en (x, y) . (Chan)



El thresholding cuenta con ciertos casos especiales, si T depende de:

- Solo $f(x, y)$, corresponde a un thresholding global.
- Ambos, $f(x, y)$ y $p(x, y)$, corresponde a un thresholding local.
- (x, y) , corresponde a un thresholding dinámico.

Un thresholding multinivel es en general menos confiable ya que es difícil establecer umbrales efectivos para aislar regiones de interés.

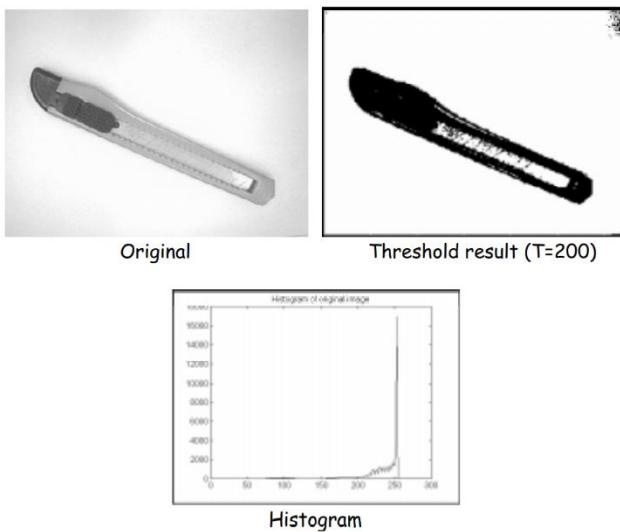


Imagen 3.23 Resultado de un thresholding no adaptativo.

Tomado de Chan

La segunda técnica de similitud es el crecimiento de región. Este es una técnica que agrupa píxeles o subregiones en regiones aún más grandes según criterios de crecimiento predefinidos. La agregación de píxeles comienza con un conjunto de punto de ‘semilla’ y, a partir de estas regiones de crecimiento, se agrega a cada ‘semilla’ los píxeles vecinos que tienen propiedades similares, como rangos de intensidad, textura o color. (González & Woods, 2018)

En la siguiente imagen se muestran distintas etapas durante el proceso de crecimiento de región.

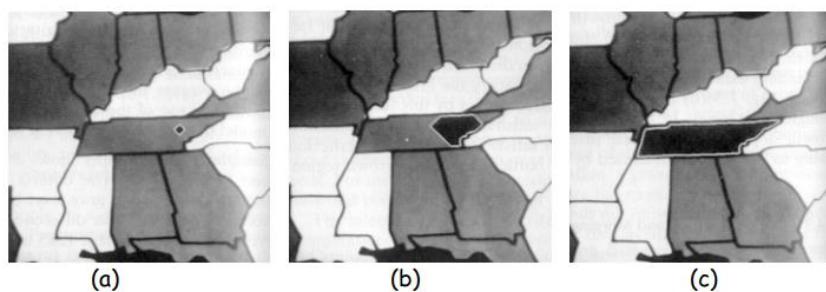


Imagen 3.24 (a) Imagen original con un punto de ‘semilla’; (b) Etapa temprana del Crecimiento de Región; (c) Región final. Tomado de Chan

Puntos a tomar en consideración al usar esta técnica es la selección de ‘semillas’ iniciales que representen adecuadamente las regiones de interés, así como la selección de propiedades adecuadas para incluir puntos en las distintas regiones durante el proceso de crecimiento. Otro punto a tomar en cuenta es la formulación de una regla de detención.

Finalmente, la última técnica es la división y fusión de región y esta alternativa subdivide una imagen inicialmente en un conjunto de regiones disjuntas y luego fusiona y/o divide las regiones en un intento de satisfacer las condiciones de segmentación.

Sea R la región de una imagen completa y Q un predicado. Un enfoque para segmentar R es subdividirlo sucesivamente en cuadrantes cada vez más pequeños, de tal modo que para todo R_i , $Q(R_i) = \text{Verdadero}$.

Teniendo a la región R , si $Q(R_i) = \text{Falso}$ para cualquier cuadrante, se subdivide ese cuadrante en otros subcuadrantes, y así sucesivamente. La forma de representación de esta técnica se llama quadtree, un árbol en el que cada nodo tiene cuatro descendientes cuyo nombre es quadregion o quadimage.

Tras haber realizado pura división dentro de la imagen, la partición final normalmente contiene regiones adyacentes con propiedades idénticas, por lo que para la segmentación se cumpla, se requiere que se fusionen solo las regiones adyacentes cuyos píxeles combinados satisfacen el predicado Q , es decir, dos regiones adyacentes R_j y R_k se fusionen solo si $Q(R_j \cup R_k) = \text{Verdadero}$.

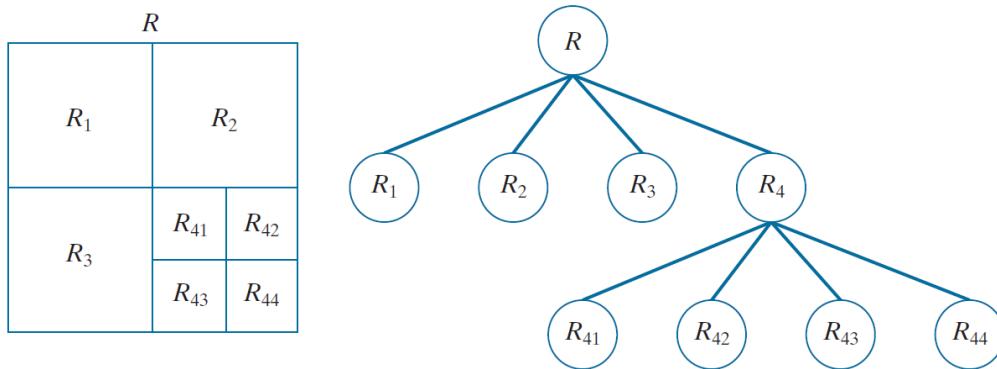


Imagen 3.25 Imagen R dividida con su respectivo quadtree. Tomado de González & Woods (2018)

3.4 Python

(Python) es un lenguaje de programación interpretado, orientado a objetos y de alto nivel con semántica dinámica. Sus estructuras de datos integradas de alto nivel, combinadas con tipado dinámico y vinculación dinámica, hacen a Python atractivo para el desarrollo rápido de aplicaciones (Rapid Application Development). También es atractivo por ser un lenguaje scripting o glue para conectar componentes existentes.

Python posee una sintaxis simple y es muy fácil de aprender, por lo que esto facilita la legibilidad y ocasionando que se reduzca el costo de mantenimiento. Asimismo, Python admite módulos y paquetes, fomentando así la modularidad del programa y la reutilización del código. Cabe mencionar que Python posee una extensa biblioteca estándar que se puede distribuir libremente.

Algunas de las aplicaciones más comunes que se le pueden dar a Python son:

- *Desarrollo Web e Internet*, por ejemplo en Frameworks como Django o micro-frameworks como Flask.
- *Computación científica y numérica*, por ejemplo para la resolución de problemas matemáticos, científicos e ingenieriles o el análisis y modelado de datos.
- *Educación*, debido a que es excelente para enseñar programación, tanto a nivel introductorio como a nivel avanzado.
- *Interfaces gráficas de usuario (GUIs) de escritorio*.
- *Desarrollo de software*, ya que Python se comporta como un lenguaje de soporte, para control y administración de compilaciones, pruebas y muchas otras maneras.
- *Aplicaciones de Negocios*, como la creación de sistemas de planificación de recursos empresariales (ERP) y de comercio electrónico (e-commerce).

Python se distribuye con una biblioteca estándar que es muy amplia y contiene módulos incorporados, escritos en C, los cuales brindan acceso a funcionalidades del sistema, así como módulos escritos en Python que proveen soluciones para los diversos problemas del día a día en la programación. Además de la biblioteca estándar, existe el Índice de Paquetes de Python

(PyPI) y es un repositorio de software para Python que nos ayuda a encontrar e instalar software desarrollado y compartido por la comunidad de Python.

En PyPI se encuentran módulos y programas individuales, hasta paquetes y frameworks completos para el desarrollo de aplicaciones.

Un framework y una librería pueden ser muy parecidos, ambos son códigos escritos por otra persona con la finalidad de ayudar a otras en la realización de tareas comunes de una manera menos detallada.

Generalmente, un framework le dice al desarrollador lo que necesita y este tiene una forma correcta de hacer las cosas, proporcionando herramientas para ser de su ayuda. En contraparte, una librería no lo hace, estas son construidas con un propósito en especial, por lo que el desarrollador llama a esta en donde y cuando se necesite. En otras palabras, el código tiene el control cuando usa una librería, pero cuando usa un framework, este último tiene el control. Esto se conoce como "inversión de control".

(Wozniewicz, 2019) usa a una casa como una metáfora para ejemplificar estos dos conceptos. Para él, una librería es como ir a Ikea. Uno ya cuenta con una casa, pero se necesita ayuda para amueblarla, sin embargo, uno no quiere hacer los muebles desde cero. Por tal motivo, Ikea te permite elegir entre distintas opciones para la casa. En este caso uno mismo tiene el control.

Por otro lado, un framework es como construir una casa modelo. Se tienen ya un conjunto de planos y algunas limitadas opciones para escoger de la arquitectura y el diseño. En última instancia, el contratista y los planos tienen el control, por lo que solo estos nos informarán cuándo y dónde uno puede dar su opinión.

Para el desarrollo de código en IA se necesitan de igual manera librerías y frameworks especiales para su desarrollo, los cuales se mencionarán algunos en los siguientes apartados.

3.4.1 Numpy

(NumPy) es una librería fundamental para la computación científica. Esta librería proporciona un objeto de matriz multidimensional, varios objetos derivados, como matrices; y una variedad de rutinas para operaciones rápidas en matrices.



Imagen 2.26 Logo de NumPy

Algunos estudios de caso importantes usando esta librería fueron la obtención de la primera imagen de un agujero negro, el Messier 87; o la confirmación de la existencia de las ondas gravitacionales por los científicos de LIGO.

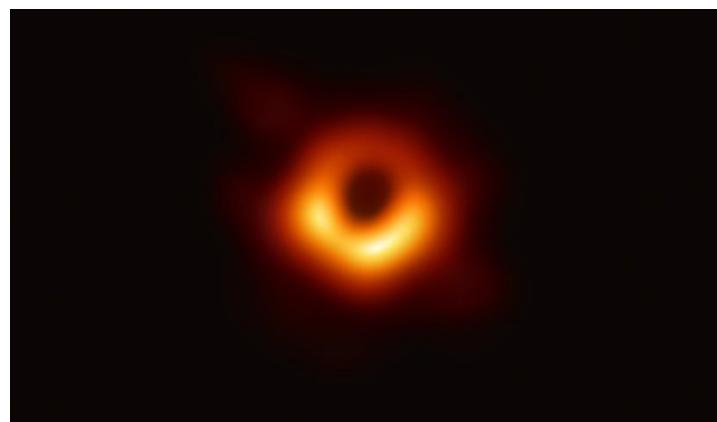


Imagen 3.27 Hoyo Negro M87. Tomado de Event Horizon
Telescope (EHT) (2019)

3.4.2 Pandas

(pandas) es una librería de código abierto que proporciona estructuras de datos rápidas, flexibles y expresivas, diseñadas para que sea fácil e intuitivo el trabajo con datos relacionales o etiquetados.

Su principal objetivo es ser el bloque de construcción fundamental de alto nivel para realizar análisis de datos prácticos del mundo real en Python.



Imagen 3.28 Logo de pandas

3.4.3 Scikit-Learn

(scikit-learn) es una librería para el Aprendizaje Automático construido sobre SciPy, un software de código abierto para matemáticas, ciencias e ingeniería.

Este proyecto fue iniciado en 2007 por David Cournapeau como un proyecto del Verano del Código de Google (Google Summer of Code), y desde entonces han contribuido muchos voluntarios. De hecho, actualmente es mantenido por un equipo de voluntarios.

Sus seis principales categorías son la clasificación, la regresión, el clustering o agrupamiento, la reducción de dimensionalidad, la selección del modelo y el preprocesamiento.



Imagen 3.29 Logo de scikit-learn

3.4.4 Matplotlib

(matplotlib) es simplemente una librería muy completa para crear visualizaciones estáticas, animadas e interactivas. Con solo pocas líneas de código se pueden desarrollar gráficos de calidad de publicación y se pueden personalizar totalmente, obteniendo así un control de los estilos de línea, propiedades de la fuente y ejes, entre otros elementos. La filosofía de matplotlib es el de hacer las cosas fáciles, sencillas y las difíciles, posibles.



Imagen 3.30 Logo de matplotlib

3.4.5 Seaborn

(seaborn) es una librería de visualización basada en matplotlib. Esta librería proporciona una interfaz de alto nivel para dibujar gráficos estadísticos bastante atractivos.

A diferencia con matplotlib, seaborn intenta hacer que un conjunto bien definido de cosas difíciles también sea fácil. seaborn ayuda a resolver dos grandes problemas que se tiene cuando se usa matplotlib los cuales son los parámetros predeterminados y el trabajo con dataframes o marco de datos.



Imagen 3.31 Logo de seaborn

3.4.6 Os

(os) se incluye en la biblioteca estándar de Python y esta proporciona funciones para interactuar con el sistema operativo. Con esta librería se puede leer o escribir en un archivo, crear archivos y directorios temporales, o solo leer todas las líneas en todos los archivos en la línea de comando.

La librería (os.path) incluye algunas funciones útiles en nombres de ruta de archivos, como por ejemplo la manipulación de estos. Los parámetros de ruta se pueden pasar como strings o bytes. También esta librería recomienda a las aplicaciones que traten de representar a los nombres de los archivos como caracteres de cadena Unicode, para evitar que algunos nombres no se puedan representar como cadenas en Unix.

3.4.7 Glob

La librería (glob) se encarga de encontrar todos los nombres de rutas de archivo que se asemejen a un patrón determinado de acuerdo a las reglas que se siguen en la terminal de Unix, aunque los resultados retornados tengan un orden arbitrario.

3.4.8 Pathlib

(pathlib) ofrece clases que representan rutas del sistema de archivos con semántica apropiada para diferentes sistemas operativos. Las clases de ruta se dividen entre rutas puras, rutas que

proporcionan operaciones puramente computacionales sin E/S; y rutas concretas, que son subclases de las rutas puras, pero estas sí proporcionan operaciones de E/S.

En esta librería se encuentra Path, cuya función consiste en crear una instancia de una ruta concreta para la plataforma en donde se está ejecutando el código.

3.4.9 OpenCV

Open-Source Computer Vision Library (OpenCV) es una librería de software de visión computacional y Aprendizaje Automático de código abierto que se creó para proporcionar una infraestructura común para aplicaciones de visión computacional y para acelerar el uso de la percepción computacional en los productos comerciales.

La librería cuenta con más de 2500 algoritmos optimizados que se pueden utilizar para detectar y reconocer rostros, identificar objetos, extraer modelos 3D de objetos, unir imágenes para producir una imagen de alta resolución de una escena completa, encontrar imágenes similares en una base de datos de imágenes, entre muchas más aplicaciones.



Imagen 3.32

Logo de OpenCV

3.4.10 TensorFlow

(TensorFlow) es una plataforma de código abierto de extremo a extremo para el Aprendizaje Automático (ML). Es ideal para una fácil y rápida creación de modelos de AA, sin importar la experiencia previa que se tenga. TensorFlow posee muchas funcionalidades diferentes, pero se destaca principalmente en redes neuronales y el Aprendizaje Profundo (DL).

TensorFlow fue desarrollado en primer lugar por investigadores e ingenieros que trabajan en el equipo de Google Brain, dentro de la organización de Investigación de IA de Google.

TensorFlow se puede ejecutar en diferentes tipos de procesadores, desde GPU y CPU en la nube y de escritorio, hasta pequeños dispositivos móviles o de IoT y la Web.

Algunos casos de éxito usando TensorFlow son:

- *GE Healthcare*, para identificar anomalías en resonancias magnéticas del cerebro.
- *PayPal*, para mantenerse a la vanguardia en la detección de fraudes.
- *Sinovation Ventures*, para la detección de enfermedades en imágenes por tomografía de coherencia óptica de la retina.
- *Texas Instruments*, para la inferencia de ML en el perímetro.
- *Twitter*, para la clasificación de tweets.



Imagen 3.33 Logo de
TensorFlow

3.4.11 Keras

(Keras) es una API de Aprendizaje Profundo (DL) que se ejecuta sobre TensorFlow. Fue desarrollado con el enfoque de permitir una experimentación rápida, ya que una clave para hacer una buena investigación es la capacidad de ir de la idea al resultado lo más rápido posible. Actualmente es usado en compañías como YouTube o Waymo, y en organizaciones científicas como el CERN, la NASA, el NIH, y muchas otras alrededor del mundo.

Keras es:

- *Simple*, pero no simplista, reduciendo la carga cognitiva al desarrollador para que este se concentre en los problemas que sí importan.
- *Flexible*, adoptando el principio de divulgación progresiva de la complejidad.
- *Potente*, proporcionando un rendimiento y una escalabilidad sólidos en la industria.



Imagen 3.34 Logo de Keras

3.4.12 PyTorch

(PyTorch) es una plataforma de investigación de Aprendizaje Profundo que ofrece flexibilidad y rendimiento y también un reemplazo de NumPy para usar la potencia de las GPUs. Algunas aplicaciones exitosas de IA haciendo uso de PyTorch son el Autopilot de Tesla y el Pyro de Uber. (Ecosystem Tools)

Algunas compañías o universidades que usan PyTorch son:

- *Salesforce*, impulsando el estado del arte en el procesamiento de lenguaje natural (NLP) y Aprendizaje Multitarea.
- *Stanford University*, para investigar de manera eficiente nuevos enfoques algorítmicos.
- *Udacity*, educando a los próximos innovadores de IA usando esta herramienta.



Imagen 3.35 Logo de PyTorch

3.4.13 Otros Frameworks

Algunos otros frameworks especializados en el Aprendizaje Profundo son:

- (Theano, 2007) desarrollado por MILA, el Instituto de Montreal para los Algoritmos de Aprendizaje.
- (Caffe, 2013) desarrollado por BVLC, el Centro de Visión y Aprendizaje de Berkeley.
- (MXNet, 2015) desarrollado por Apache Software Foundation.
- (Microsoft Cognitive Toolkit (CNTK), 2016) desarrollado por Microsoft Research.
- (Deeplearning4j, 2014) desarrollado por el equipo de Ingeniería del startup Skymind. Este framework es de código abierto y para JAVA.
- (Chainer, 2015) desarrollado por Preferred Networks Inc.

3.5 Herramientas

Para el desarrollo del código de un modelo de Aprendizaje Profundo se necesitan ciertas herramientas para llevarlo a cabo de manera satisfactoria, como los son el lugar de donde se va a obtener el conjunto de datos a entrenar, el lugar donde se va a construir o desarrollar el modelo y el lugar donde se va a almacenar el proyecto.

3.5.1 Kaggle

(Kaggle) es un subsidiario de Google LCC y es la comunidad de ciencia de datos y Aprendizaje Automático más grande del mundo. La plataforma de Kaggle permite comenzar un nuevo proyecto de ciencia de datos de la forma más rápida posible, simplemente se hace uso de kernels en un Jupyter Notebook hospedada en la nube, y de un conjunto de datos gratuito de los más de 100,000 disponibles dentro del repositorio de la plataforma.

En Kaggle se puede buscar o publicar conjuntos de datos, buscando que la plataforma sea para fines educativos y brinde la oportunidad de que sirvan de práctica para los entusiastas de aprender sobre la ciencia de datos.

Sin embargo, Kaggle también permite a los científicos de datos y desarrolladores en participar en concursos, fomentando de esta forma un entorno competitivo en donde se brindan premios en efectivo a los ganadores. Al ser Kaggle un sitio de calidad y confianza, las empresas como Facebook, Walmart y Winton Capital, publican proyectos o problemas a los que se enfrentan en tiempo real para que los Kagglers puedan resolverlos y tengan la oportunidad de ganar productos gratuitos o posiblemente una oferta de trabajo.

Uno de los proyectos más grandes y reconocidos es el de Heritage Health, proyecto que ofreció \$3 millones como premio.

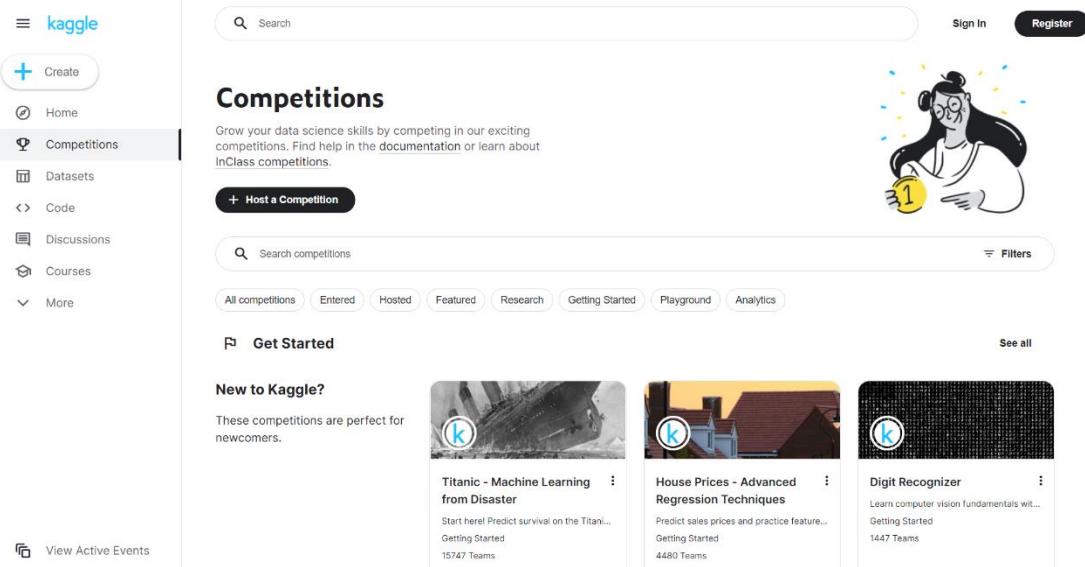


Imagen 3.36 Sitio Web Kaggle en la sección de competiciones

3.5.2 Jupyter Notebook

Un Jupyter Notebook, o un cuaderno Jupyter, es una aplicación web de código abierto que permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo. Algunos de sus usos incluyen la limpieza y transformación de datos, la simulación numérica, el modelado estadístico, la visualización de datos, el Aprendizaje Automático y muchas más aplicaciones. (Project Jupyter)

La interfaz de usuario de un Jupyter Notebook contiene el nombre del cuaderno, la barra de menú, la barra de herramientas y celdas de código. El formato en que se guardan los Jupyter Notebook es ‘.ipynb’.

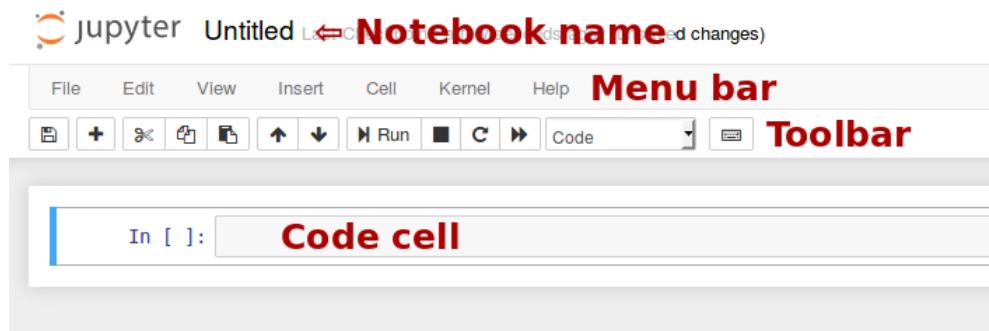


Imagen 3.37 Interfaz de usuario de un documento de Jupyter Notebook

Un Jupyter Notebook cuenta con tres tipos de celda distintos:

- *Celdas de código*, permiten escribir y editar código nuevo con resaltado completo de sintaxis y autocompletado con el tabulador. El lenguaje de programación que usa depende del kernel y el kernel predeterminado (IPython) ejecuta código Python.
- *Celdas de nota*, permiten el texto enriquecido y el formato de código en HTML y su principal función es enfatizar el texto dentro del cuaderno.
- *Celdas sin procesar*, proporcionan un lugar en el que se puede escribir la salida directamente. El cuaderno no evalúa estas celdas y cuando se pasan a través de la función ‘nbconvert’, las celdas sin procesar llegan sin modificación alguna al formato de destino.

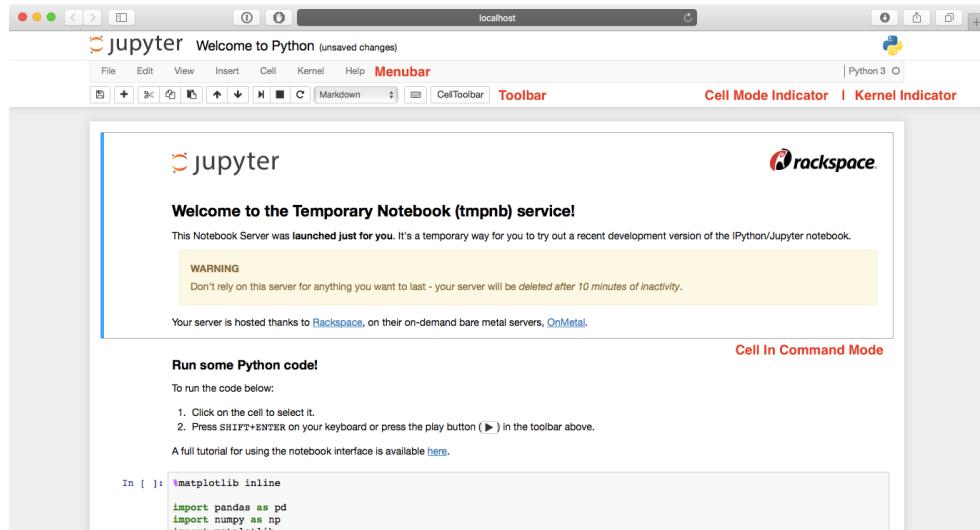


Imagen 3.38 Editor de un Jupyter Notebook

3.5.3 Google Colaboratory

Google Colaboratory o Google Colab es un documento ejecutable que permite escribir, ejecutar y compartir código dentro de Google Drive. Un símil es que Google Colab es como un Jupyter Notebook, pero almacenado en Google Drive.

Colaboratory es gratuito por lo que simplemente se necesita una cuenta de Google para que todo trabajo o proyecto realizado quede almacenado en el Google Drive personal.

Un Notebook está compuesto de celdas y cada una de estas puede contener código ejecutable, texto enriquecido, imágenes, HTML, LaTeX y muchos más elementos. Una de sus mayores ventajas es que es muy sencillo compartir los Notebooks, permitiendo que sean fáciles de editar.

Colab funciona conectando el Notebook a un entorno de ejecución en la nube, logrando así la ejecución de código de Python sin necesidad de una configuración previa en un dispositivo local, siendo esta la diferencia contra los Jupyter Notebooks. También Google da acceso gratuito a GPUs. (Google LLC)

```
print('Hello world!')
```

Hello world!

Imagen 3.39 Colaboratory Notebook con una celda que imprime ‘Hello World!’ y su respectiva respuesta de salida

3.6 Microsoft Azure

Azure es la plataforma de informática en la nube de Microsoft con un conjunto de servicios en la nube en expansión constante que ayudan a la organización a crear soluciones para cumplir los desafíos empresariales actuales y futuros. Azure le ofrece la libertad de compilar, administrar e implementar aplicaciones en una red global masiva mediante sus herramientas y plataformas favoritas. (Microsoft Corporation)

Azure admite infraestructura, plataforma y software como informática de servicio, con servicios como máquinas virtuales que se ejecutan en la nube, alojamiento de servicios web y bases de datos, y servicios informáticos avanzados como IA, ML e IoT.

Los servicios de Aprendizaje Automático e Inteligencia Artificial de Azure brindan a los desarrolladores y científicos de datos una amplia gama de experiencias productivas para crear, entrenar e implementar modelos de Aprendizaje Automático de una manera más rápida.

Azure ofrece una basta cantidad de servicios informáticos en la nube en los que la gestión de la infraestructura, la escalabilidad, la disponibilidad y la seguridad son gestionadas por uno mismo, permitiendo así que se ahorre tiempo y dinero. La mayoría de los servicios de Azure son pay-as-you-go, es decir, que solo se pagan por el tiempo de cómputo que se utilizaron.

En la Imagen 3.40 se observan algunos de los servicios que ofrece Microsoft Azure en sus respectivas categorías, como lo son los servicios de proceso, redes, almacenamiento, móvil, bases de datos, web, IoT, Big Data, IA y DevOps.

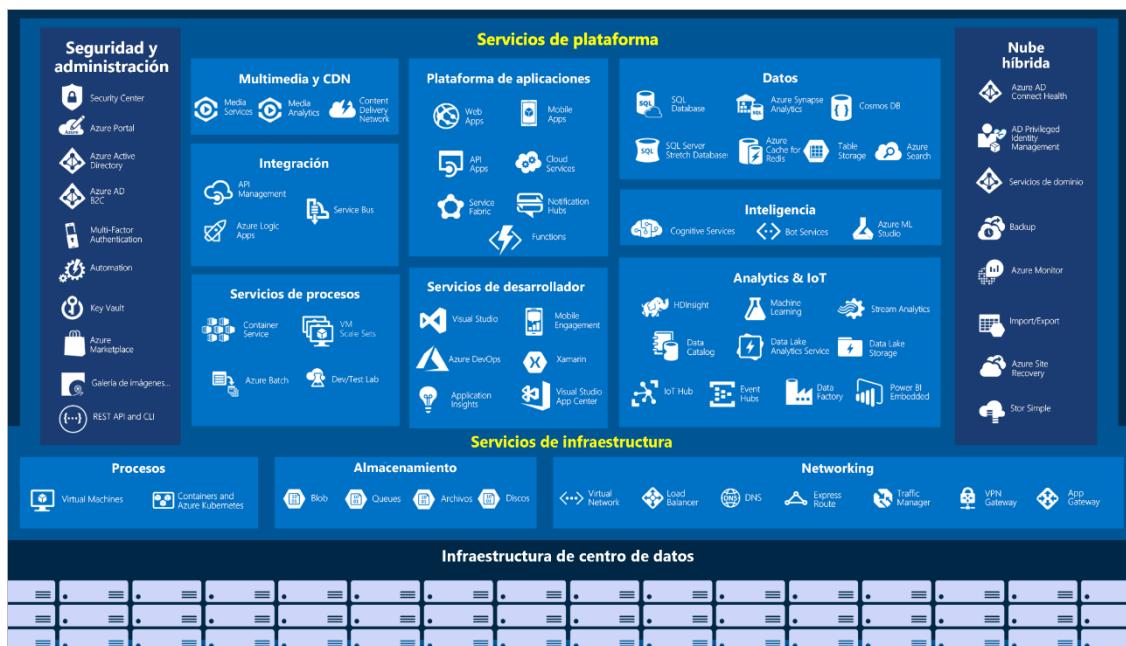


Imagen 3.40 Vista general de los servicios y características disponibles en Microsoft Azure

3.6.1 Azure Portal

Azure Portal es un sitio web donde se pueden crear, administrar y supervisar todos los servicios de Azure disponibles, siendo así una solución integral de administración gráfica.

Con Azure Portal, se puede administrar la suscripción de Azure mediante una interfaz gráfica de usuario (GUI) donde se pueden crear paneles personalizados para una vista organizada de los recursos, facilitando así la compilación, administración y supervisión de todo, desde el alojamiento de una aplicación web sencilla hasta implementaciones complejas en la nube.

El Azure Portal no deja de actualizarse y no requiere tiempo de inactividad para las actividades de mantenimiento.

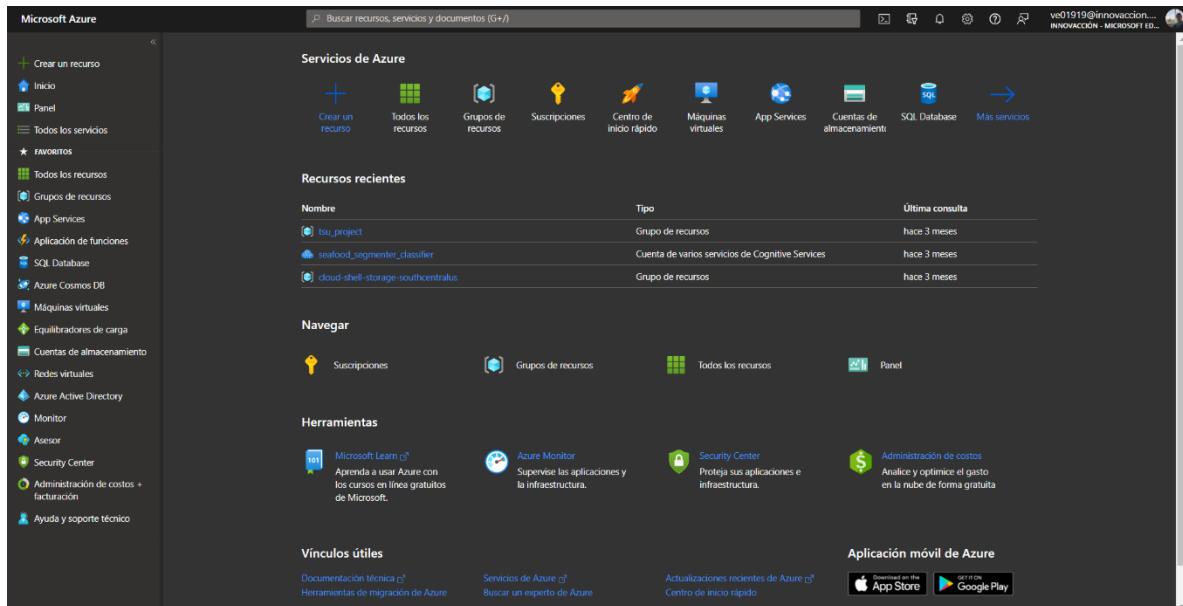


Imagen 3.41 Página de inicio de Azure Portal

3.6.2 Azure Marketplace

Azure Marketplace permite a los clientes buscar, probar, comprar y aprovisionar aplicaciones y servicios de cientos de los principales proveedores de servicios, todos ellos certificados para ejecutarse en Microsoft Azure. (Microsoft Corporation)

El catálogo de soluciones abarca varias categorías del sector, incluyendo, pero sin limitarse a plataformas de contenedores de código abierto, imágenes de máquina virtual, bases de

datos, software de compilación e implementación de aplicaciones, herramientas para desarrolladores, detección de amenazas y cadena de bloques. Con Azure Marketplace, se puede aprovisionar soluciones integrales de forma rápida y confiable, hospedadas en un entorno propio de Azure.

Aunque Azure Marketplace está diseñado para profesionales de TI y desarrolladores de nube interesados en software de TI y comercial, los asociados de Microsoft también lo usan como un punto de inicio para todas las actividades conjuntas de comercialización.

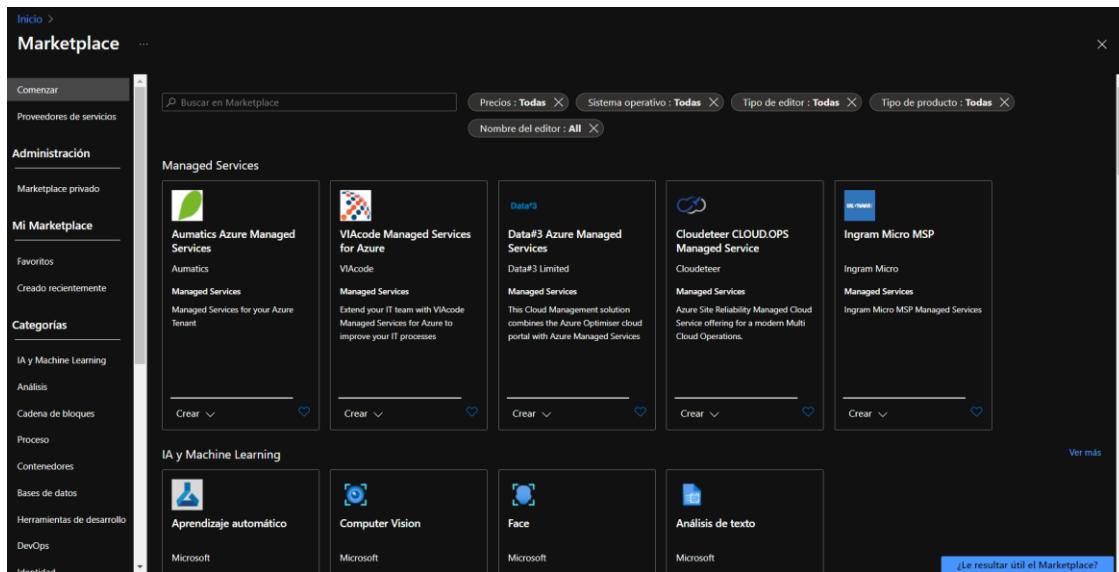


Imagen 3.42 Página de inicio de Azure Marketplace

3.6.3 Microsoft Cognitive Services

Microsoft Cognitive Services son servicios basados en la nube con APIs REST y SDK (Kit de Desarrollo de Software) de biblioteca de cliente disponibles para ser de ayuda en el desarrollo de inteligencia cognitiva en aplicaciones. (Microsoft Corporation)

Una API REST, o API RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a las limitaciones del estilo arquitectónico REST y permite la interacción con los servicios web RESTful. REST significa transferencia de estado representacional y fue creado por el científico de la computación Roy Fielding. (Red Hat, 2020)

Un SDK es un conjunto de herramientas de creación de software para una plataforma específica, incluidos los componentes básicos, los depuradores y, a menudo, un framework como un conjunto de rutinas específicas de un sistema operativo. (IBM, 2021)

Microsoft Cognitive Services pone a la IA al alcance de todos los desarrolladores, sin necesidad de que tengan habilidades o conocimientos de Aprendizaje Automático, solo basta una llamada API para incrustar funciones cognitivas, como la capacidad de ver, oír, hablar, buscar, entender y acelerar la toma de decisiones en las aplicaciones.

Cognitive Services se categoriza en cuatro áreas principales:

- *Visión*, para identificar y analizar el contenido de las imágenes y los videos.
- *Voz*, para integrar el procesamiento de voz en aplicaciones y servicios.
- *Idioma*, para extraer significado de texto no estructurado.
- *Decisión*, para hacer decisiones más inteligentes con mayor rapidez.

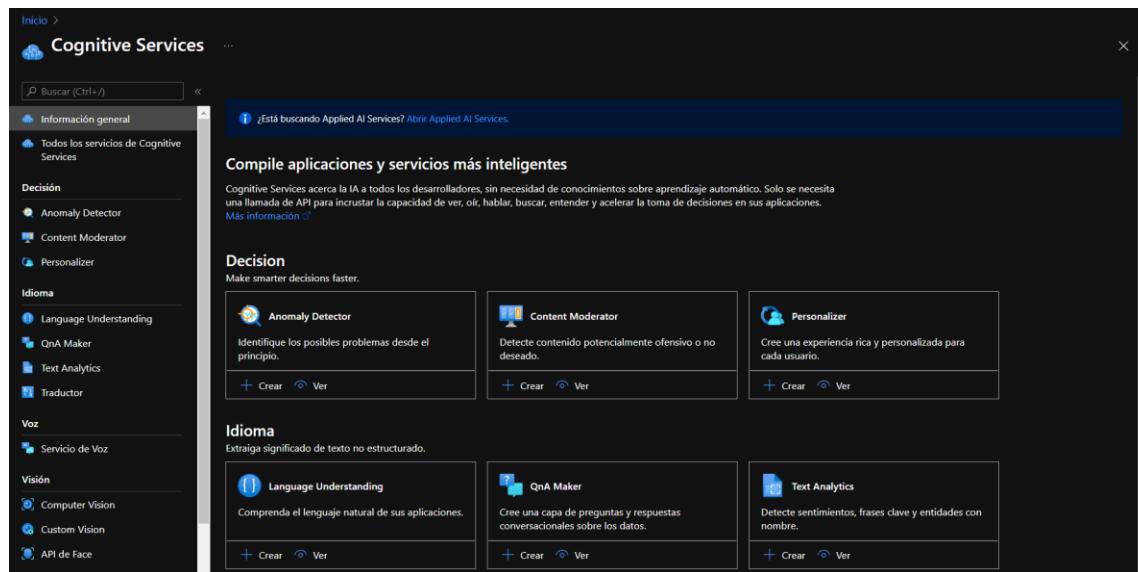


Imagen 3.43 Pantalla de inicio de Microsoft Cognitive Services

3.6.4 Custom Vision

Azure Custom Vision es un servicio de reconocimiento de imágenes que permite crear, implementar y mejorar modelos de identificador de identificador de imagen. Un identificador de imagen aplica etiquetas a las imágenes, de acuerdo con las características visuales

detectadas. A diferencia del servicio de Azure Computer Vision, Custom Vision permite especificar etiquetas propias y entrenar modelos personalizados para que estas sean detectadas. Esto se logra ya que utiliza un algoritmo de Aprendizaje Automático para analizar imágenes. (Microsoft Corporation)

La funcionalidad de Custom Vision se divide en dos funciones:

- *Clasificación de imágenes*, donde se aplica una o más etiquetas a una imagen.
- *Detección de objetos*, la cual es similar, pero también devuelve las coordenadas en la imagen donde se pueden encontrar las etiquetas aplicadas.

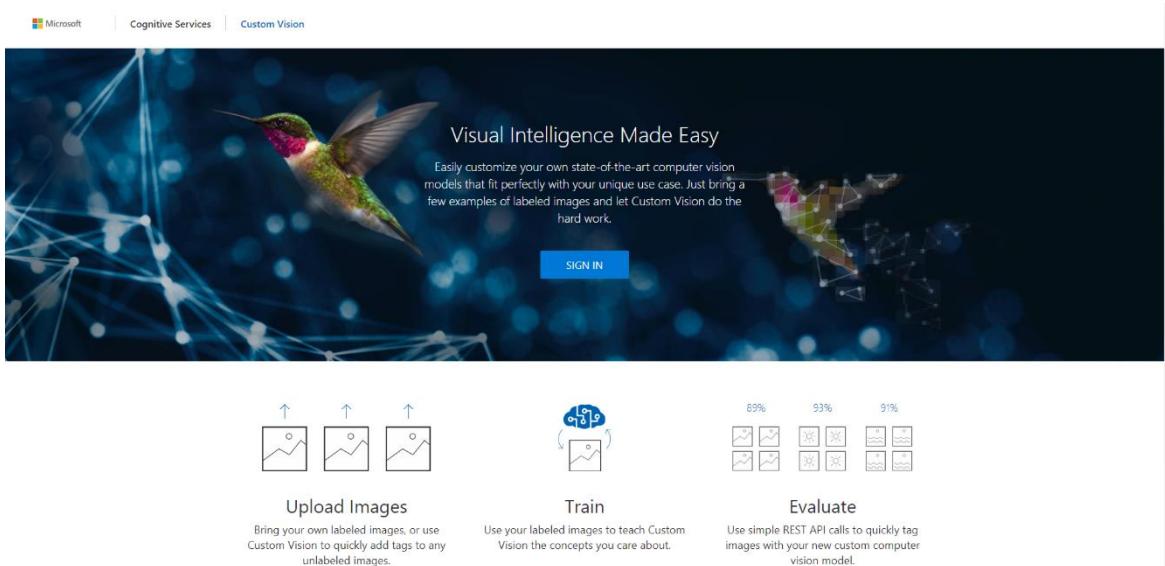


Imagen 3.44 Página de inicio de Custom Vision

3.7 GitHub

(GitHub) es una interfaz basada en web donde se puede cargar una copia de un repositorio de Git. Fue desarrollado por Chris Wanstrath, P. J. Hyett, Tom Preston-Werner y Scott Chacon usando Ruby on Rails en el 2007.

Actualmente es el mayor proveedor de alojamiento de repositorios y muchos proyectos de código abierto lo utilizan para hospedar su Git, realizar su seguimiento de fallos, hacer sus revisiones de código, entre muchas otras funciones.

GitHub permite colaborar de una manera más sencilla con otras personas dentro de un mismo proyecto, debido a que el proyecto está en una ubicación central para compartir el repositorio, se tiene una interfaz basada en web para visualizarlo y herramientas como el forking, los pull requests, los issues, y los wikis, permitiendo así que se logre especificar, discutir y revisar los cambios realizados por el equipo de una manera más eficiente. (Bell & Beer, 2015)

En general, las ventajas que trae GitHub son:

- Se puede documentar bugs o especificar nuevas funciones que se quieren realizar, con la herramienta *issues*.
- Se puede colaborar en distintas ramas haciendo uso de *ramas* y *pull requests*.
- Se puede revisar los últimos cambios realizados en el repositorio con un *pull request*.
- Se puede ver el progreso del equipo en el repositorio a través del historial de *commits*.

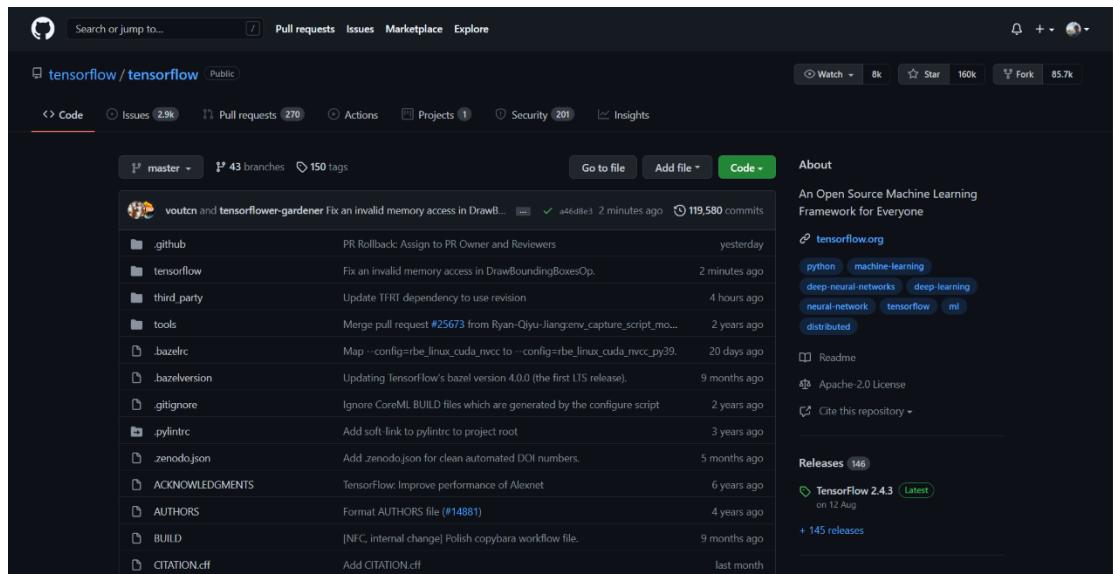


Imagen 3.45 Ejemplo de un repositorio en GitHub. Tomado de TensorFlow

3.7.1 Git

(Git) es un sistema de control de versiones. Un sistema de control de versiones es una herramienta de software diseñada con el propósito de realizar un seguimiento de los cambios que se realizan en los archivos a lo largo del tiempo.

Git fue desarrollado por Linus Torvalds y originalmente fue diseñado para operar en un entorno Linux, aunque actualmente es multiplataforma. (Bell & Beer, 2015)

En general, las ventajas de Git son:

- Se puede deshacer los cambios si llega a haber un error, recuperando una versión anterior del trabajo.
- Se puede consultar una versión anterior del proyecto para ver exactamente cuál era el estado de los archivos en un tiempo determinado en el pasado.
- Se puede documentar los cambios realizados y sirven como una futura referencia.
- Se puede tener la confianza de realizar cualquier cambio ya que siempre se puede volver a una versión anterior del trabajo.
- Se puede crear diferentes ramas para experimentar para desarrollar nuevas funciones de forma independiente, para después llegarlos a fusionar con la rama principal del proyecto, o simplemente descartarlos y eliminarlos sin llegar a ser fusionados.

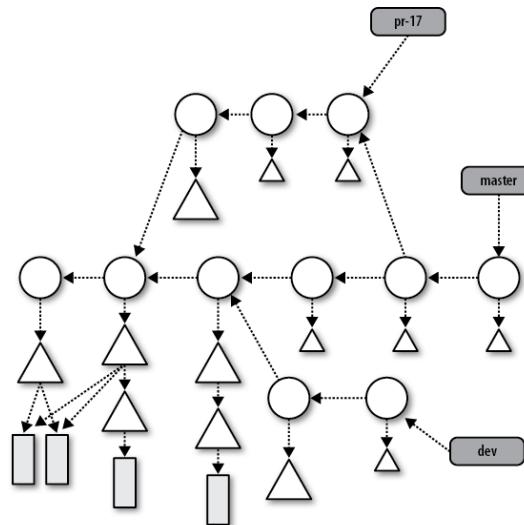


Imagen 3.46 Gráfico completo de commits. Tomado de Loeliger & McCullough (2012)

CAPÍTULO 4

DESARROLLO DEL PROYECTO

4.1 Introducción al Entorno de Trabajo

Para la realización de este proyecto, se debe tener un editor de texto donde se puedan crear archivos que soporten el lenguaje de programación en el que se desarrolla el proyecto, en este caso Python.

Se puede ocupar un editor de texto para trabajar los archivos localmente, el que más recomiendo es Visual Studio Code debido a que es gratuito, está disponible para Windows, MacOS y Linux; es extensible y personalizable ya que en él se pueden trabajar con una infinidad de lenguajes distintos; tener distintos temas, depuradores y muchos otros servicios adicionales, como funciones de Azure. También cuenta con comandos de Git integrados.

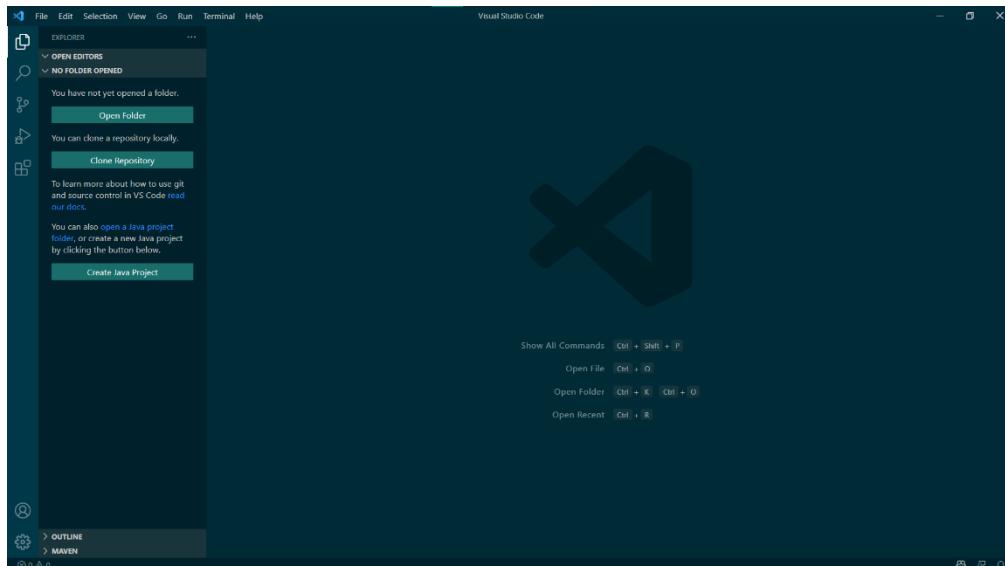


Imagen 4.1 Ventana de inicio de Visual Studio Code

Sin embargo, Google Colab es la plataforma seleccionada para el desarrollo de los algoritmos del modelo de Aprendizaje Profundo, ya que nos permite ejecutar el proyecto en la nube y

almacenarlo en un Google Drive personal, así como tener un control de versiones de este y la posibilidad de añadirlo en un repositorio público en GitHub.

4.1.1 Introducción a Google Colaboratory

Colab no necesita una configuración previa para ser utilizado, solamente se necesita tener una cuenta de Google e iniciar sesión. Con el siguiente enlace se puede acceder a Google Colaboratory: <https://colab.research.google.com/>.

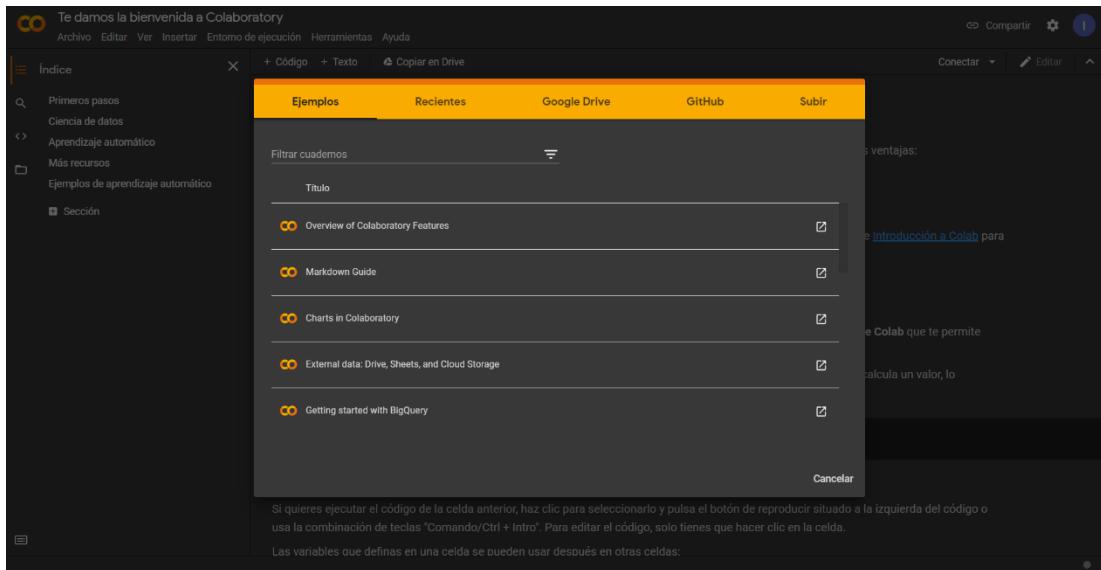


Imagen 4.2 Página principal de Google Colaboratory

Ya en la página principal podemos acceder a distintas pestañas en las que podemos abrir distintos tipos de cuadernos, o Notebooks, ya sea abrir cuadernos de Ejemplos, cuadernos Recientes, cuadernos almacenados en el Google Drive de la misma cuenta con la que se inició sesión, cuadernos almacenados en un repositorio en GitHub o simplemente cargar uno desde una computadora local.

Al hacer clic en *Cancelar* en la ventana que se muestra en la Imagen 4.2, se puede observar un Notebook titulado “Te damos la bienvenida a Colaboratory”, en el cual se brinda una introducción a la plataforma y al uso de los Notebooks, así como su uso en Ciencia de Datos o en Aprendizaje Automático. De igual forma, en el Notebook se pueden encontrar con más recursos y ejemplos.

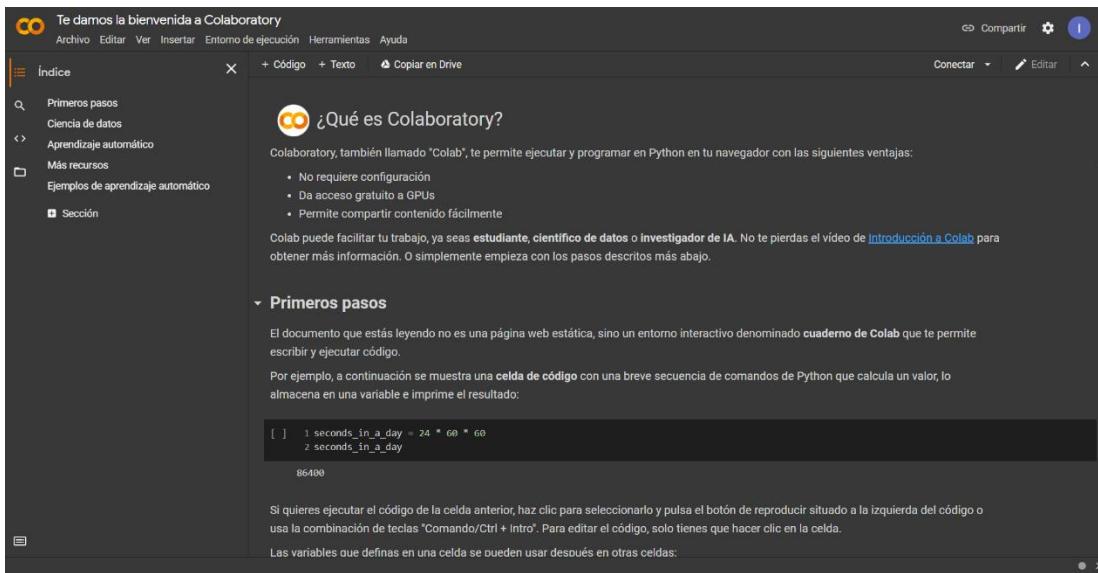


Imagen 4.3 Notebook "Te damos la bienvenida a Colaboratory"

En la pestaña *Archivo* se pueden crear nuevos cuadernos, cargar o descargar cuadernos, así como guardar copias en un repositorio público de GitHub.

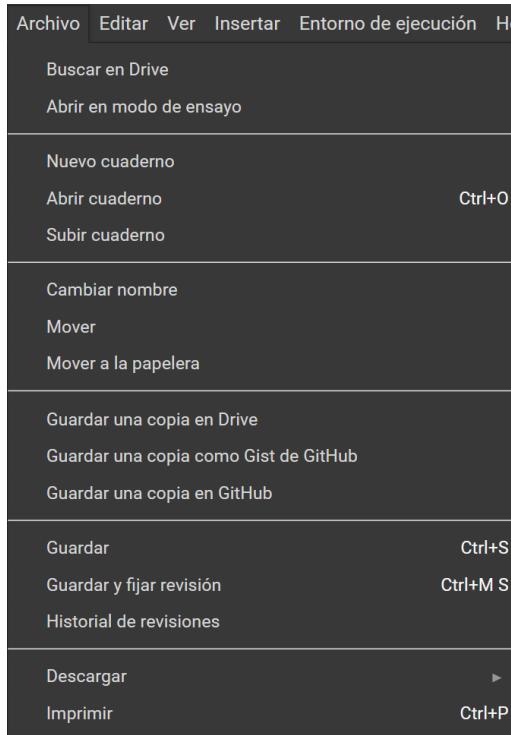


Imagen 4.4 Elementos de la pestaña *Archivo*

En la pestaña *Entorno de ejecución*, al hacer uso del elemento *Gestionar sesiones* se pueden gestionar las sesiones activas, mientras que con el elemento *Cambiar tipo de entorno de ejecución* se puede acceder a GPUs o TPUs disponibles para brindar aceleración por hardware a cálculos de larga duración.

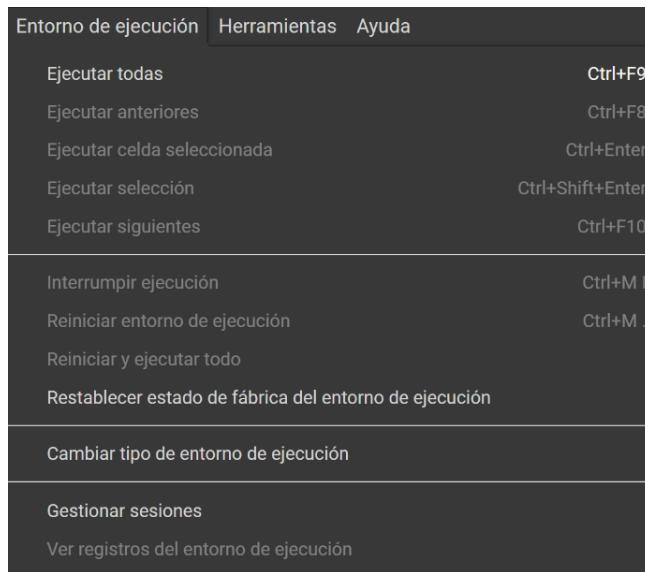


Imagen 4.5 Elementos de la pestaña *Entorno de ejecución*

Finalmente, en la parte superior derecha, se encuentra la pestaña *Conectar*, pestaña en la que brinda distintas opciones para conectarse, ya sea en entornos de ejecución alojados en la nube o localmente

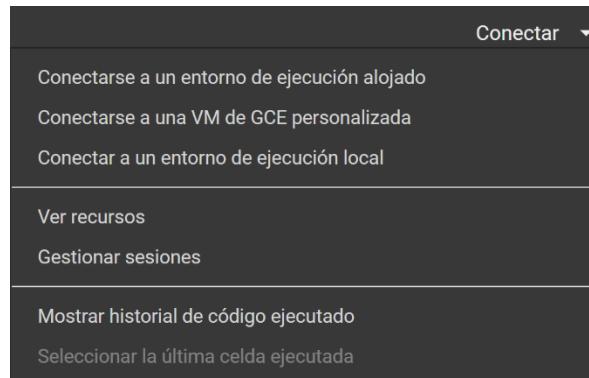


Imagen 4.6 Elementos de la pestaña *Conectar*

4.1.2 Introducción a Kaggle

Para acceder a un Jupyter Notebook alojado en la nube a través de la plataforma Kaggle, primero se debe crear una cuenta en <https://www.kaggle.com/>.

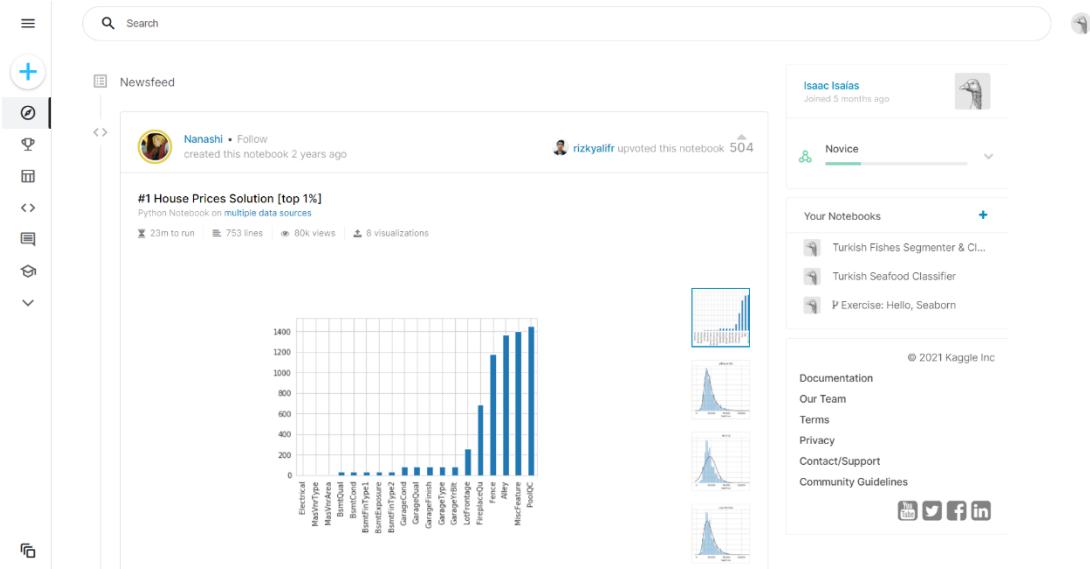


Imagen 4.7 Página principal de Kaggle

Después de iniciar sesión en Kaggle, al estar ya en la plataforma, se observa en la parte izquierda del sitio web una barra de menús, que tiene la opción de contraerse, en donde se puede acceder a distintas funciones. Dentro de estas funciones, al hacer clic en el menú con el símbolo +, se pueden crear un conjunto de datos o un Jupyter Notebook. Seleccionando *New Notebook* se crea un Jupyter Notebook.

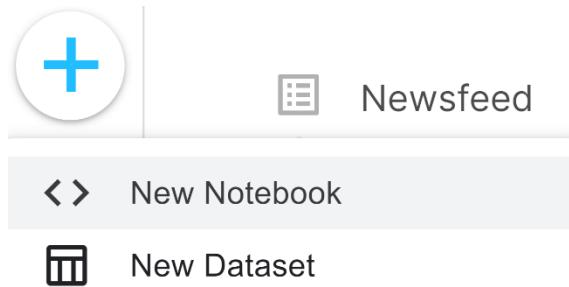


Imagen 4.8 Opciones del menú +

Al crearse el Jupyter Notebook, esta automáticamente se guardará en la cuenta; del lado derecho del entorno de trabajo se cuenta con un menú en donde se puede seleccionar el lenguaje de programación, Python o R, así como la opción de escoger un acelerador de hardware alojado en la nube, ya sea una GPU o una TPU v3-8.

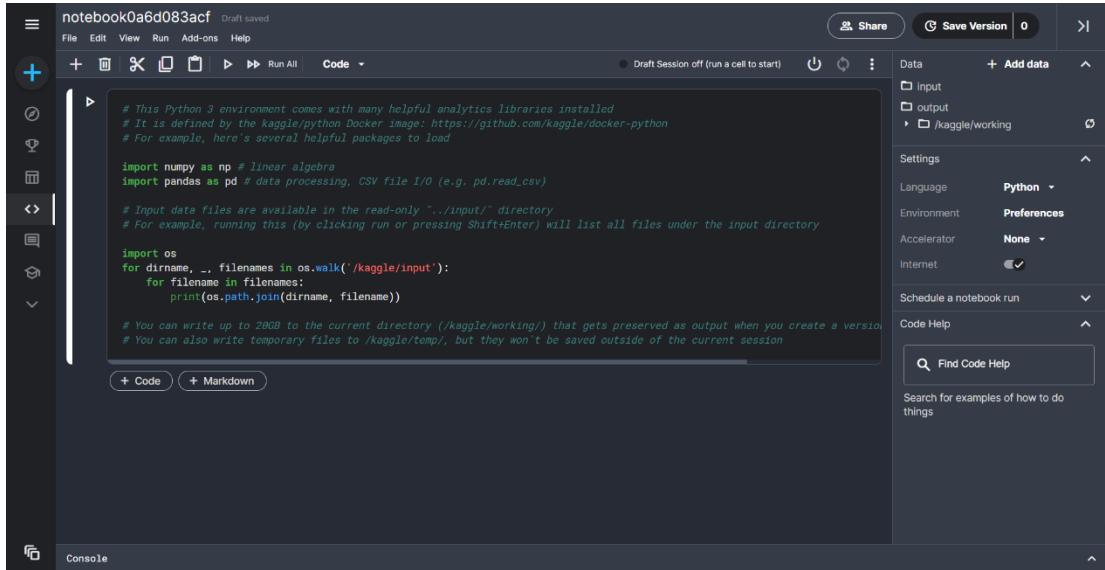


Imagen 4.9 Un Notebook recién creado en Kaggle

Cabe mencionar que también se puede realizar un control de versiones de los Notebooks, para esto se selecciona *Save Version* en la parte superior derecha del entorno de trabajo del Notebook.

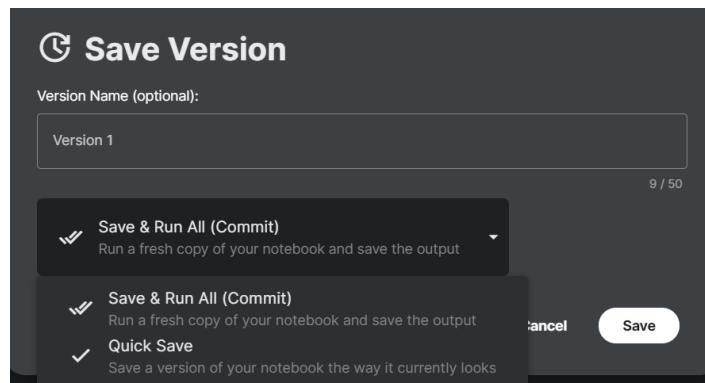


Imagen 4.10 Menú *Save Version*

4.2 Principios de Python

Python 3, por su sencillez y su fácil aprendizaje, es el lenguaje destinado para el desarrollo rápido del modelo de Aprendizaje Profundo con redes neuronales.

Para reforzar los conocimientos de Python, se tomaron cursos de distintas plataformas y se leyó la documentación oficial del sitio web de Python. Los cursos que sirvieron para reforzar este lenguaje interpretado son:

- Python Core de SoloLearn
- Curso Profesional de Python de CódigoFacilito

Python Core de SoloLearn consta de diez módulos, que abarca desde conceptos básicos hasta pitonicidad y empaquetamiento. Cada módulo consta de lecciones y cada lección consta de ejercicios teóricos y prácticos. Al final de cada uno de los módulos, se hace un cuestionario general de este y un proyecto de codificación para poner en práctica los conocimientos adquiridos. Finalmente, al resolver los diez proyectos de codificación se obtiene el certificado.

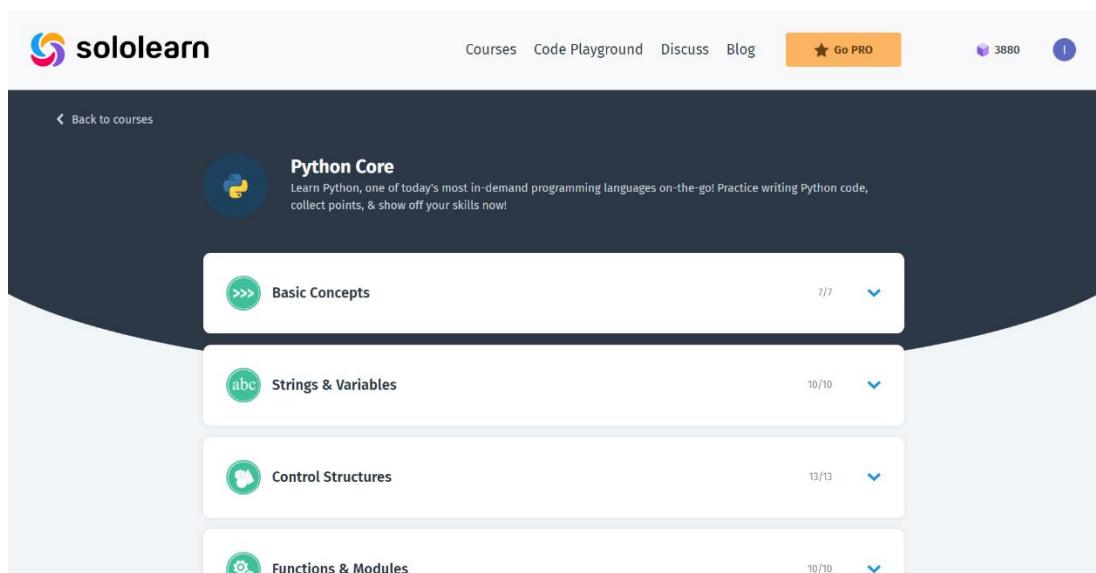


Imagen 4.11 Página del curso de Python Core en SoloLearn

El Curso Profesional de Python de CódigoFacilito se especializa en tratar más a fondo los ciclos y condiciones, las funciones y las clases; pero en especial las estructuras de los distintos tipos de datos.

Imagen 4.12 Página del Curso Profesional de Python en CódigoFacilito

Como en todo lenguaje de programación, en Python también se usan variables para almacenar los distintos tipos de datos. Por convención de la comunidad Python, se recomienda usar la notación snake case (`snake_case`) para nombrarlas y también se recomienda que posean un nombre descriptivo para que sean fáciles de reconocer. Un par de ejemplos de declaración de variables son:

```
[1] 1 primer_nombre = 'Isaac'
     2 costo_boleto = 7500
```

Imagen 4.13 Ejemplos de declaración de variables

En Python no existen las constantes ya que sus valores pueden ser cambiados, sin embargo existe una convención para definirlas y esta es nombrándolas con la notación screaming snake case (`SCREAMING_SNAKE_CASE`); además de usar esa notación para nombrarlas,

se sugiere que estas variables solo se usen para lectura. Un par de ejemplos de declaración de variables como constantes son:

```
[2] 1 PI = 3.141592
2 RAZON_AUREA = (1 + (5 ** (1/2))) / 2
```

Imagen 4.14 Ejemplos de declaración de constantes

Una de las razones por las que Python no tiene constantes es porque es un lenguaje de tipado dinámico, esto significa que tiene la posibilidad de que las variables puedan almacenar diferentes tipos de datos en diferentes momentos de la codificación. Esta propiedad agiliza el proceso de desarrollo del código, pero un inconveniente es que existe la posibilidad que se esto genere un código difícil de leer.

Para evitar tener un código difícil de leer, se recomienda documentar el código de forma adecuada haciendo uso de comentarios. Para comentarios de una línea de código se usa el numeral (#), mientras para comentarios de varias líneas se utilizan triple dobles comas al inicio y al final del comentario. Algunos ejemplos son:

```
[3] 1 # Este es un simple comentario
2
3 '''
4 Este es un
5 comentario de
6 varias líneas
7 de código.
8 '''
```

Imagen 4.15 Ejemplos de comentarios

Los distintos tipos de datos que se pueden almacenar en variables son:

- String: Cadenas de texto entre comillas simples o dobles comillas; ejemplos: ‘Isaac’ y “Isaías”.
- Int: Representa a todos los números enteros, ya sean negativos, positivos o el cero; ejemplos: -27, 0 y 8.

- Float: Representa a todos los números de punto flotante o los números decimales; ejemplos: -27.7, 0.0 y 8.4.
- Bool: Dato booleano y tiene dos estados, True o False.

Los operadores matemáticos se usan con los datos de tipo *int* o *float* para efectuar cálculos matemáticos, estos son la suma (+), la resta (-), la multiplicación (*), la exponenciación (**), la división (/), el cociente de una división a través de la función piso (//) y el residuo de una división haciendo uso del operador módulo (%).

Además de los operadores matemáticos, Python también usa operadores relacionales y operadores lógicos.

Los operadores relacionales están diseñados para comparar tipos de datos numéricos. Estos operadores son:

- >, mayor
- <, menor
- >=, mayor e igual
- <=, menor e igual
- ==, igual
- !=, diferente

Los operadores lógicos están diseñados para comparar tipos de datos booleanos. Estos operadores son:

- *and*
- *or*
- *not*

Las estructuras de datos en Python están muy bien diseñadas, ya que estas permiten organizar los datos de tal manera que se puedan almacenar colecciones de datos y realizar operaciones o hacer relaciones entre ellos, por lo que la resolución de cualquier problema es posible. Las breves explicaciones de cada una de las estructuras de datos son:

- List (listas): Las listas son dinámicas porque pueden aumentar o decrecer su tamaño en tiempo de ejecución. A la posición de una lista se le llama índice y con él se obtienen y se actualizan datos dentro de la lista. Una propiedad interesante con las listas es que se pueden formar matrices. Para crear una lista se usan corchetes ([]) y dentro de ellos se separan los elementos con una coma; o se usa la función list() y como argumento un objeto iterable. Algunos ejemplos son:

```
[4] 1 list_1 = [0, 1, 2, 3, 4, 5]
2 print(list_1)
3 type(list_1)
```

```
[0, 1, 2, 3, 4, 5]
list
```

```
[5] 1 list_2 = list((2, 3, 5, 7))
2 print(list_2)
3 type(list_2)
```

```
[2, 3, 5, 7]
list
```

Imagen 4.16 Ejemplos de listas de datos

- Tuple (tuplas): La principal característica de las tuplas es que son inmutables, es decir, que no pueden modificar los valores definidos desde el principio. Se pueden utilizar las mismas funciones que las de las listas, con excepción de las funciones con las que se tengan la intención de modificar las tuplas, sino generará un error. Para crear una tupla se usan paréntesis (()) y dentro de ellos se separan los elementos con una coma; o se usa la función tuple() y como argumento un objeto iterable. Algunos ejemplos son:

```
[6] 1 tuple_1 = (2, 4, 6, 8, 10)
2 print(tuple_1)
3 type(tuple_1)

(2, 4, 6, 8, 10)
tuple

[7] 1 tuple_2 = tuple([-5, -4, -3, -2, -1])
2 print(tuple_2)
3 type(tuple_2)

(-5, -4, -3, -2, -1)
tuple
```

Imagen 4.17 Ejemplos de tuplas de datos

- Dictionary (diccionarios): Son estructuras de datos mutables y no se rigen por índices, sino por llaves. En los diccionarios, todo valor necesita una llave; y toda llave necesita un valor. Los diccionarios se ordenan como fueron creados y mantienen el último valor al que son asignados. Para crear un diccionario se usan llaves ({}), las llaves se relacionan con los valores a través de dos puntos (:) y cada par de llave con su valor se separa con una coma; o se usa la función dict() y como argumento los pares de llave y su valor. Algunos ejemplos son:

```
[8] 1 dict_1 = { 'firstName' : 'Isaac', 'lastName' : 'López', 'age' : 19}
2 print(dict_1)
3 type(dict_1)

{'firstName': 'Isaac', 'lastName': 'López', 'age': 19}
dict

[9] 1 dict_2 = dict(country='Mexico', state='Querétaro', city='San Juan del Río')
2 print(dict_2)
3 type(dict_1)

{'country': 'Mexico', 'state': 'Querétaro', 'city': 'San Juan del Río'}
```

Imagen 4.18 Ejemplos de diccionarios de datos

- Set (conjuntos): Almacenan datos de manera desordenada y sin elementos duplicados. Los objetos dentro de los conjuntos admiten operaciones matemáticas como la unión (|), la intersección (&), la diferencia (-) y la diferencia simétrica (^). Para crear un conjunto, se pueden usar llaves ({}) o no, solo deben estar los elementos separados por una coma; o se usa la función set() y como argumento un objeto iterable. Algunos ejemplos son:

```
[10] 1 set_1 = {'maths', 'physics', 'biology', 'chemistry', 'maths'}
2 print(set_1)
3 type(set_1)

{'biology', 'physics', 'maths', 'chemistry'}
set

[11] 1 set_2 = set('Python')
2 print(set_2)
3 type(set_2)

{'P', 'h', 'o', 'y', 'n', 't'}
set
```

Imagen 4.19 Ejemplos de conjuntos de datos

En Python se puede usar la palabra reservada *if* para hacer condiciones cuyo resultado esperado sea verdadero. En cambio, la palabra reservada *else* es usada en condiciones esperadas como falsas. Al usar *if* y *else* es necesario delimitar con una indentación de cuatro espacios las sentencias dentro de estas palabras para delimitar los bloques de código; pero en Colab solo es necesario hacer uso de dos espacios o un tab. Un ejemplo de su uso es el siguiente:

```
[12] 1 number = 17
2 if number == 10:
3 | print('El número es igual a 10.')
4 else:
5 | print('El número no es igual a 10')

El número no es igual a 10
```

Imagen 4.20 Ejemplo de una condición *if/else*

Otra funcionalidad importante en Python como en todos los lenguajes de programación son los bucles o ciclos. Los bucles utilizados en Python son el bucle *for* y el bucle *while*.

El bucle *for* es capaz de iterar sobre cada uno de los elementos dentro de un objeto iterable.

Un ejemplo es el siguiente:

```
[13] 1 componentes = ['LED', 'res', 'cap', 'bob', 'pot']
      2 for componente in componentes:
      3     print(componente)

      LED
      res
      cap
      bob
      pot
```

Imagen 4.21 Ejemplo de ciclo *for*

Un bucle *while* permite ejecutar n veces un bloque de código hasta que una condición se cumpla. Un ejemplo es el siguiente:

```
[14] 1 contador = 1
      2 while contador <= 5:
      3     print(contador)
      4     contador += 1

      1
      2
      3
      4
      5
```

Imagen 4.22 Ejemplo de ciclo *while*

Si tanto en los bucles *for* y *while* no se usa una condición, no es posible generar un bloque de código con *else*. Además, dentro del bloque de código del bucle *while*, es posible ocupar dos palabras reservadas especiales que pueden modificar el comportamiento de este; con *break* se puede finalizar inmediatamente el ciclo, mientras que con *continue* se puede saltar a la siguiente iteración.

4.2.1 Principios de NumPy

La librería NumPy es usada para trabajar con datos numéricos, ya que esta incluye una diversa cantidad de funciones y estructuras de datos para realizar una amplia variedad de operaciones matemáticas. Para comenzar a utilizar NumPy, primero se necesita importarlo y renombrarlo como *np* para que sea más práctico trabajar con la librería.

```
[15] 1 import numpy as np
      2 np.__version__
      '1.19.5'
```

Imagen 4.23 Importación y versión de NumPy

En Python, las listas son usadas para almacenar datos, sin embargo, NumPy provee una estructura más rápida y más compacta que estas. El objeto se llama ndarray y son matrices multidimensionales de elementos del mismo tipo y tamaño; estas se crean usando la función *np.array()*.

```
[16] 1 np.array([range(n, n + 3) for n in [1, 4, 7]])
      array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])
```

Imagen 4.24 Ejemplo de una matriz multidimensional NumPy

Algunas manipulaciones básicas que se le pueden hacer a las matrices son:

- La determinación del tamaño, la forma, el consumo de memoria y los tipos de datos en las matrices.
- La obtención y establecimiento del valor a elementos individuales de una matriz.
- La obtención y configuración de submatrices más pequeñas dentro de una matriz más grande.
- El reestructuramiento de la forma de una matriz dada.
- La combinación de varias matrices en una, y la división de una matriz en muchas.

NumPy provee una cuantiosa cantidad de funciones universales (ufuncs) entre ellas los conocidos operadores aritméticos y relacionales; las funciones trigonométricas y trigonométricas hiperbólicas; así como estadísticas de resumen como la media y la desviación estándar o funciones agregadas como la suma, el producto, la mediana, el mínimo y el máximo.

4.2.2 Principios de Manipulación de Datos con Pandas

Pandas al ser construido sobre NumPy, comparte muchas funciones y propiedades de esta. Con Pandas se puede leer y extraer datos de archivos, transformarlos y analizarlos, calcular estadísticas y correlaciones, y muchas otras funciones. Para utilizar esta librería, primero se necesita importarlo y renombrarlo como *pd* por convención, más que nada para que sea más práctico trabajar con la librería.

```
[17] 1 import pandas as pd
2 pd.__version__
'1.1.5'
```

Imagen 4.25 Importación y versión de Pandas

En Pandas, se proveen dos estructuras principales para facilitar el trabajo con los datos, estos son las *Series* y los *DataFrames*. A su vez, en estas estructuras también se encuentra otra estructura de Pandas llamada *Index*.

Las *Series* proveen una estructura de datos para matrices de un solo tipo, como las matrices de NumPy. Más que nada, son esencialmente una columna.

Los *DataFrames* se basan en las columnas *Series*, solo que estas pueden tener muchas más columnas, como si fuera una hoja de cálculo. En resumen, los *DataFrames* son una tabla multidimensional hecha de columnas *Series*.

La estructura *Index* prácticamente da etiquetas de fila, lo que permite hacer una selección de fila, siendo así un papel importante en la identificación de entradas de datos. Esto hace que las *Series* sean más poderosas que las matrices NumPy.

```
[18] 1 date = np.array(['06-06-2021', '20-06-2021',
2 '10-10-2021', '24-10-2021', '07-11-2021'])

[19] 1 dates = pd.Series(date, name='date')
2 dates

0    06-06-2021
1    20-06-2021
2    10-10-2021
3    24-10-2021
4    07-11-2021
Name: date, dtype: object
```

Imagen 4.26 Ejemplo de una *Serie*

```
[20] 1 data = {
2     'Date' : ['06-06-2021', '20-06-2021',
3             '10-10-2021', '24-10-2021', '07-11-2021'],
4     'Country' : ['Azerbaijan', 'France',
5                  'Turkey', 'United States', 'Mexico']
6 }
7
8 dataframe = pd.DataFrame(data, index=data['Date'])
9 dataframe.drop('Date', inplace=True, axis=1)
10 dataframe
```

	Country
06-06-2021	Azerbaijan
20-06-2021	France
10-10-2021	Turkey
24-10-2021	United States
07-11-2021	Mexico

Imagen 4.27 Ejemplo de un *DataFrame* con las fechascomo *Index* en el argumento

Por último, se tiene a una función importante para seleccionar datos en función de su índice número, esta tiene por nombre *iloc* y funciona de la misma manera como la indexación de listas en Python.

```
[21] 1 print(dataframe.iloc[1:4])
```

	Country
20-06-2021	France
10-10-2021	Turkey
24-10-2021	United States

Imagen 4.28 Ejemplo del uso la función *iloc*

4.2.3 Principios de Visualización con Matplotlib

Matplotlib puede realizar un sinfín de medios visuales, pero en el proyecto se utiliza un módulo en específico, este módulo se llama *pyplot*. Para llamar a este módulo en específico de la librería se le renombrará como *plt* por la convención que se tiene al usarlo.

```
[22] 1 import matplotlib.pyplot as plt
```

Imagen 4.29 Importación del módulo *pyplot* de *matplotlib*

El gráfico más básico y sencillo es un gráfico de líneas. En la Imagen 4.30, los valores de *x* van del 0 al 4.9 con un margen de 0.1; mientras que los valores de *y* son los valores de $\sin(x)$. Para crear el gráfico de líneas, solo se llama a la función *plot()* y como argumentos a *x* y.

```
[23] 1 x = np.arange(0, 5, 0.1)
2 y = np.sin(x)
3
4 plt.plot(x, y);
```

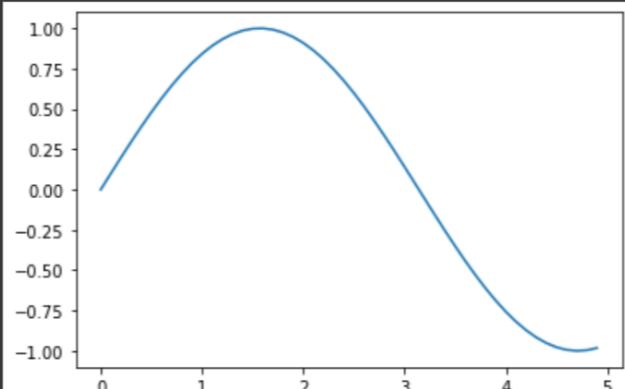


Imagen 4.30 Ejemplo de un gráfico sencillo de líneas

La API orientada a objetos se recomienda para gráficos más complejos y cuando se quiera tener un control mayor sobre las figuras. En la interfaz, las funciones de trazado son métodos de objetos explícitos de *Figure* y *Axes*. En la Imagen 4.31 se aprecia un ejemplo de esta aplicación.

```
[24] 1 x = np.linspace(0, 10, 100)
2
3 # Primero crea una cuadrícula de trama
4 # ax será una matriz de dos objetos Axes
5 fig, ax = plt.subplots(2)
6
7 # Llama al método plot() en el objeto apropiado
8 ax[0].plot(x, np.sin(x))
9 ax[1].plot(x, np.cos(x));
```

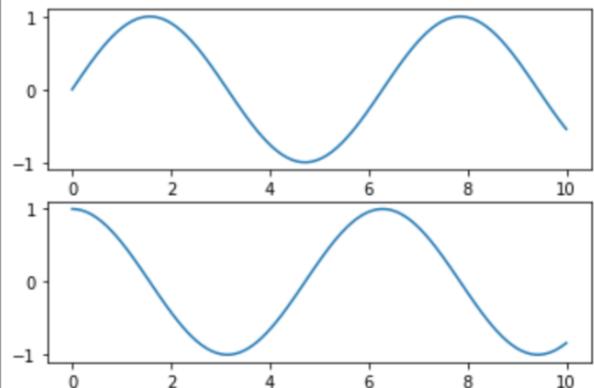


Imagen 4.31 Subtramas que utilizan la API orientada a objetos

4.3 Principios de Deep Learning

Para aprender a desarrollar modelos de Aprendizaje Profundo, o Deep Learning, con redes neuronales se tomó un curso de *Udacity* titulado *Deep Learning Nanodegree program*.

El curso consiste de seis partes, que en conjunto brindan una comprensión profunda y completa del Aprendizaje Profundo y cubren algunos de los temas más principales de esta área. Cada unidad cuenta con una gran cantidad de vídeos teóricos y Notebooks con ejercicios; los Notebooks están escritos con Python y el framework de Deep Learning utilizado es PyTorch. En el curso hay seis proyectos reales del mundo hechos por expertos de industrias Top en tecnología, dentro de las seis partes del curso; estas partes son:

1. Introducción al Deep Learning
2. Redes Neuronales
3. Redes Neuronales Convolucionales

4. Redes Neuronales Recurrentes
5. Redes Generativas Antagónicas
6. Implementación de Modelos de Aprendizaje Automático

4.3.1 Introducción al Deep Learning

En la primera parte se muestran pocas aplicaciones interesantes del Deep Learning usando modelos pre entrenados que están disponibles en GitHub, con la finalidad de dar un acercamiento del enorme potencial de esta herramienta de Inteligencia Artificial. La primera aplicación que se hace es la transferencia de estilo, cuyo objetivo es recrear imágenes o videos propios con estilos de pinturas famosas. El modelo se encuentra en el siguiente repositorio:

<https://github.com/lengstrom/fast-style-transfer.git>

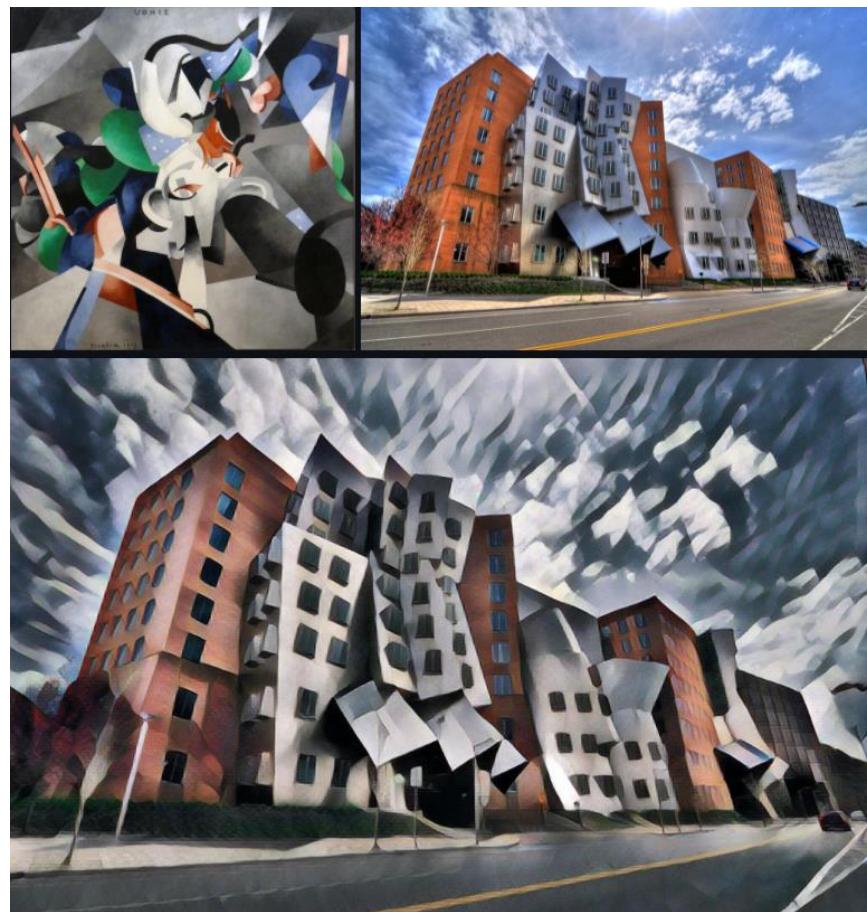


Imagen 4.32 MIT Stata Center como Udnie, por Francis Picabia

También se aprende a como crear un entorno virtual en Anaconda, entorno donde se desarrollan los Notebooks del curso; y una introducción a los Jupyter Notebooks con Python y NumPy. Además, se da una introducción a la regresión lineal y al Aprendizaje Automático, con el propósito de familiarizarse con el vocabulario necesario para entender los conceptos de las siguientes unidades, y la capacidad de saber en dónde se puede abordar el Aprendizaje Profundo dentro del Aprendizaje Automático.

4.3.2 Redes Neuronales

En la segunda parte del curso se aprende a como construir una red neuronal simple desde cero usando Python; se ven desde conceptos como el perceptrón hasta algunos algoritmos utilizados, como el descenso de gradiente y la retropropagación. Así mismo, tiene un enfoque especial con el análisis de sentimientos debido a que se desarrolla una red neuronal para procesar texto y predecir sentimientos a través de seis mini proyectos a lo largo de la unidad.

Una aplicación del análisis de sentimientos es el poder clasificar entre positivas y negativas reseñas de películas o series a través de redes neuronales. Para ello se deben transformar los textos en números, es decir, realizar un etiquetado a las palabras con 1 o 0; de tal forma que un 1 es una palabra positiva, mientras que un 0 una palabra negativa.

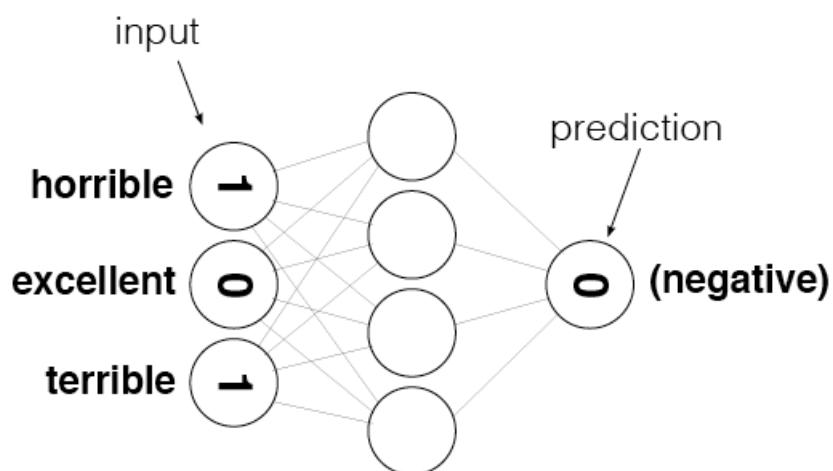


Imagen 4.33 Ejemplo con salida 0 tras detectar palabras negativas

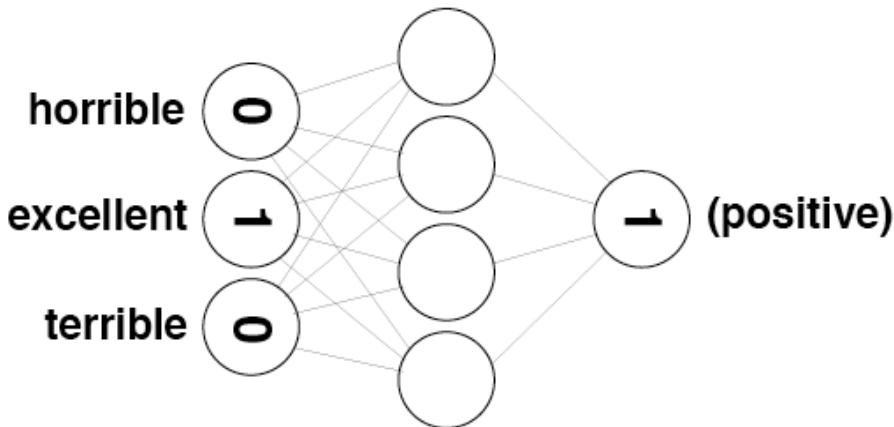


Imagen 4.34 Ejemplo con salida 1 tras detectar una palabra positiva

El primer proyecto del curso se encuentra en esta unidad y el proyecto consiste en construir y entrenar un modelo de redes neuronales que prediga el número de usuarios de bicicletas compartidas en un día determinado de un establecimiento.

4.3.3 Redes Neuronales Convolucionales

La tercera unidad es sobre las redes neuronales convolucionales; estos algoritmos se destacan en la visión computacional y procesamiento de imágenes por su capacidad de detectar e identificar rasgos importantes. Una aplicación que se hace con las redes convolucionales en el curso es la creación de un autoencoder cuya arquitectura de red es usada para la compresión y eliminación de ruido en las imágenes.

Un ejemplo de la aplicación de un autoencoder es la compresión del conjunto de datos MNIST. Con los autoencoders simples, se pasan los datos de entrada a través de un *encoder*, o codificador, que hace representación comprimida de la entrada. Luego, esta representación se pasa a través de un *decoder*, o decodificador, para reconstruir los datos de entrada. Por otro lado, los autoencoders convolucionales, su parte del *encoder* está hecha con capas convolucionales y capas de agrupamiento; mientras el *decoder* está hecho de capas convolucionales transpuestas que aprenden a muestrear una representación comprimida.

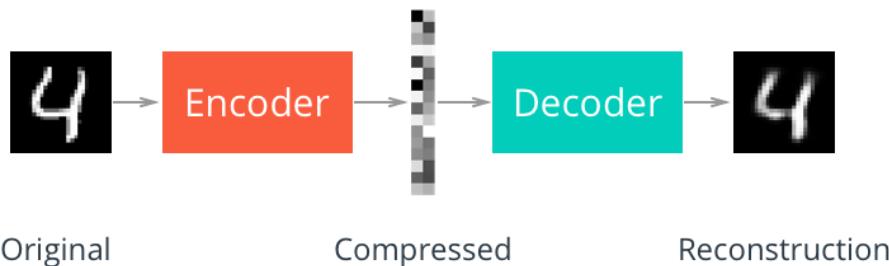


Imagen 4.35 Autoencoder del conjunto de datos MNIST

Por último en la parte teórico y práctico de la unidad, se hace uso de una red neuronal pre entrenada para clasificar imágenes en un modelo en el que jamás las haya visto.

El segundo proyecto del curso es la construcción de un modelo de redes convolucionales para una clasificación multiclas entre distintas razas de perros a través de imágenes.

El tercer proyecto del curso también se encuentra en la unidad de redes convolucionales, este proyecto es una clasificación y etiquetado de puntos de referencia para las redes sociales. La finalidad es crear un clasificador de referencia en el que los servicios para compartir o almacenar fotografías etiqueten automáticamente las fotos con marcadores de ubicación.

4.3.4 Redes Neuronales Recurrentes

La cuarta parte del curso trata sobre las redes neuronales recurrentes, redes cuya arquitectura se adecúa con datos que forman secuencias de texto, música y datos de series temporales. Estas redes utilizan secuencias como entradas en la fase de entrenamiento, y elementos de memoria. La memoria se define como la salida de las neuronas de capa oculta, neuronas que sirven como una entrada adicional a la red durante el siguiente paso de entrenamiento.

En esta unidad se aprende sobre incrustaciones de palabras y el modelo Word2Vec, red que aprende sobre las relaciones semánticas entre palabras. Estas se utilizan para aumentar la eficiencia de las redes cuando se procesa texto. En resumen, las incrustaciones y las RNN predicen el sentimiento de las críticas de películas, un gran ejemplo de las tareas comunes en el procesamiento de lenguaje natural.

El cuarto proyecto del curso es sobre la generación de guiones de TV. Este proyecto usa el framework TensorFlow para que pueda procesar el texto.

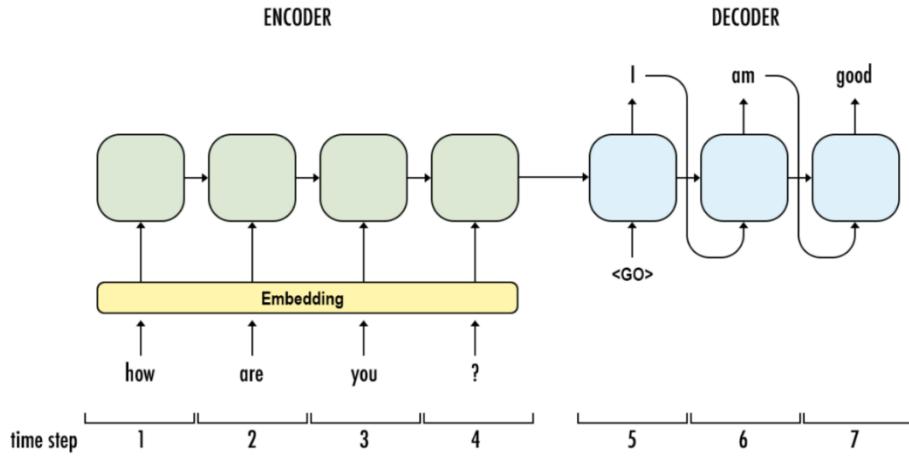


Imagen 4.36 Un ejemplo de una estructura RNN en donde un codificador representa la pregunta: "how are you?" y un decodificador genera la respuesta: "I am good"

4.3.5 Redes Generativas Antagónicas

La quinta parte del curso enseña una de las arquitecturas más nuevas del Deep Learning, estas son las redes generativas antagónicas o redes de confrontación generativa y han estado mostrando una gran capacidad de comprensión a los datos del mundo real. En esta unidad se aprende y se implementan las GANs para una diversa cantidad de tareas, así como la codificación de una CycleGAN.

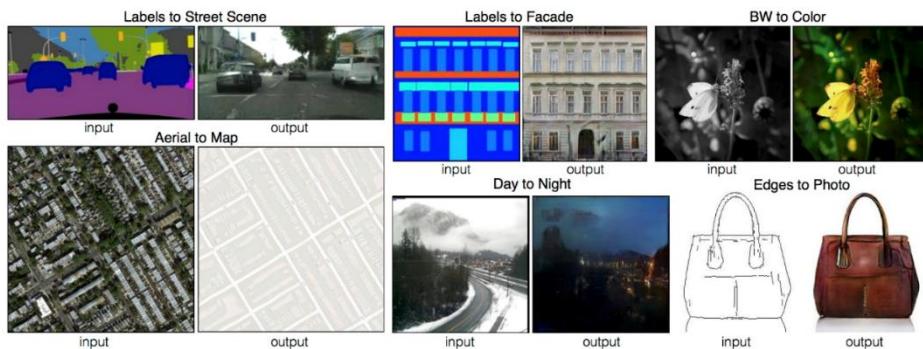


Imagen 4.37 Ejemplos de traslación imagen a imagen realizada por formulaciones CycleGAN y Pix2Pix

El quinto proyecto consiste en utilizar GANs para generar nuevas imágenes de rostros humanos lo más reales posibles.

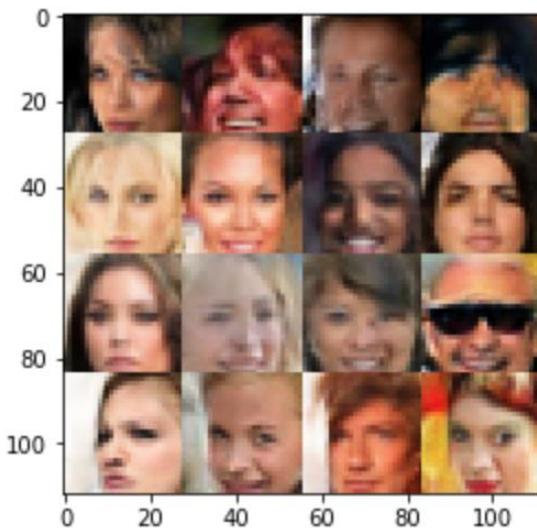


Imagen 4.38 Imágenes de baja resolución generadas por
GANs

4.3.6 Implementación de Modelos de Aprendizaje Automático

La sexta y última parte del curso consiste simplemente en obtener experiencia en la implementación de un modelo para que se pueda acceder a él a través de una aplicación web y a la vez, responder a las entradas de los usuarios.

La implementación de modelos de aprendizaje automático para audiencias globales es una realidad, ya que una basta cantidad de empresas buscan crear productos de Inteligencia Artificial, por lo que existe una creciente demanda de ingenieros que puedan realizar dicha actividad.

El sexto y último proyecto consiste en entrenar e implementar un modelo de análisis de sentimientos de PyTorch a través de una puerta de enlace creada con la finalidad de que se pueda acceder al modelo a través de un sitio web.

4.3.7 Recursos Complementarios

El curso abarca las seis partes vistas anteriormente en esta sección del capítulo, sin embargo, Udacity ofrece unas lecciones adicionales para complementar de mejor forma el curso.

La primera lección adicional consiste en un conjunto de material audiovisual en que explica la evaluación de métricas, como lo son una matriz de confusión. También se da una explicación de la precisión y la exhaustividad, conocidos en inglés como *precision* y *recall*; y los falsos positivos, y los falsos negativos.

La segunda lección adicional trata de la regresión lineal, una de las dos principales familias de algoritmos, al igual que la clasificación. Esta lección brinda un acercamiento profundo a la regresión, ya que no solo se aprende el concepto, sino que se aprenden maneras de mejorarlo y finalmente, la búsqueda de maneras de generalizar casos no lineales.

La tercera lección adicional es un módulo dedicado al aprendizaje y comprensión de gráficos diferenciables, como la retropropagación o el gradiente descendente estocástico. Esto se logra con el uso del framework TensorFlow, ya que con la abstracción fundamental para ejecutar y entrenar redes permite que se construya una pequeña librería llamada MiniFlow.

Por último, como últimos recursos complementarios, se tienen un conjunto de tres lecciones adicionales. Las tres lecciones están enfocadas en la API Keras. Para poder ocupar Keras, se necesita un backend, en este caso es TensorFlow. En general, los temas que se abarcan son una introducción a Keras, una introducción a TensorFlow y un mini proyecto donde se crea una CNN con estas dos herramientas del Deep Learning.

4.4 Principios de Git y GitHub

Todo proyecto de código debe tener su propio repositorio y este proyecto no es una excepción. Para ello, primero se debe descargar el conjunto de datos con el que se va a trabajar de la plataforma de Kaggle. El conjunto de datos con que se va a trabajar se llama *A Large Scale Fish Dataset* y se encuentra en el siguiente enlace:

<https://www.kaggle.com/crowww/a-large-scale-fish-dataset>

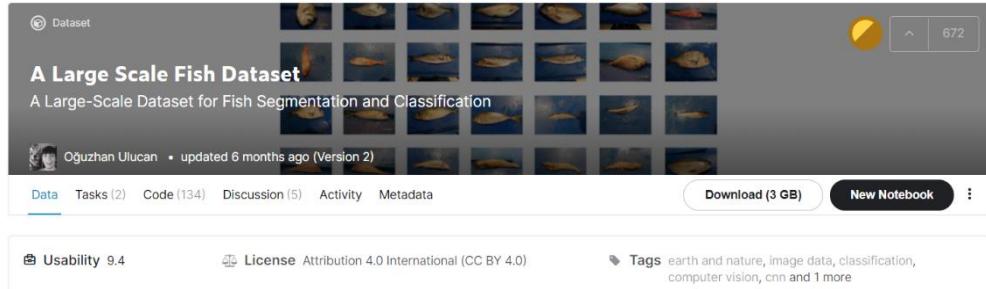


Imagen 4.39 Conjunto de datos *A Large Scale Fish Dataset* en Kaggle

A partir de este momento al término ‘conjunto de datos’ se le llamará dataset, más que nada para abreviarlo y porque suele llamarse más así en el desarrollo de proyecto de IA.

Después de descargar el dataset, este se encuentra en formato ZIP en la ruta *Descargas* del equipo local. Posteriormente, se descomprime el archivo ZIP en una carpeta. A continuación, se prosigue con la instalación de Git para poder crear el repositorio local.

4.4.1 Instalación de Git y Git Bash

Como primer paso, ingresamos al siguiente enlace, con el cual se accede al sitio web de Git:

<https://git-scm.com/>

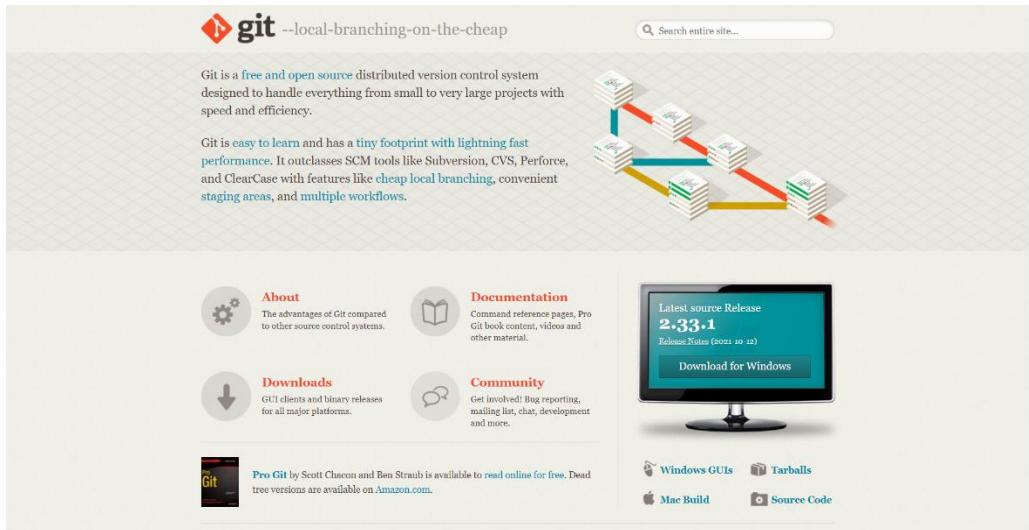


Imagen 4.40 Página de inicio de Git

Automáticamente la página web reconoce el sistema operativo de la computadora, por lo que solo basta dar clic en el botón de la imagen de la computadora aguamarina para que comience la descarga del instalador de Git. En mi caso el SO es Windows.

Después de que el instalador de Git se haya descargado, se ejecuta este como administrador. A continuación sale una ventana de información y se le da clic en *Next*. La siguiente ventana es para seleccionar la ubicación de destino donde Git se instalará.

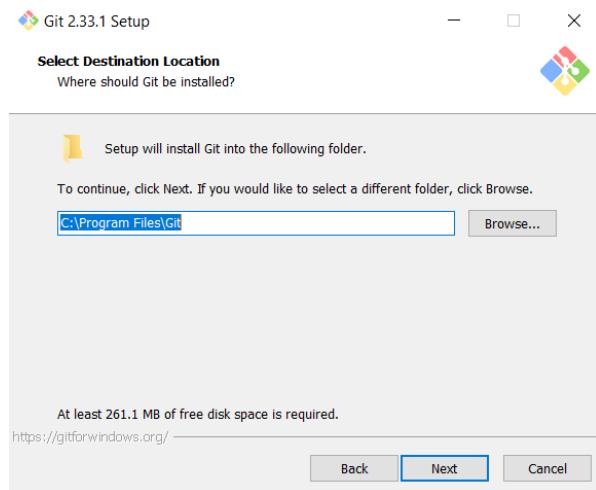


Imagen 4.41 Ventana de la ubicación de destino

Ahora, en la ventana siguiente se seleccionan los componentes que se desean instalar. Es importante que se tenga seleccionada la casilla *Git Bash Here*.

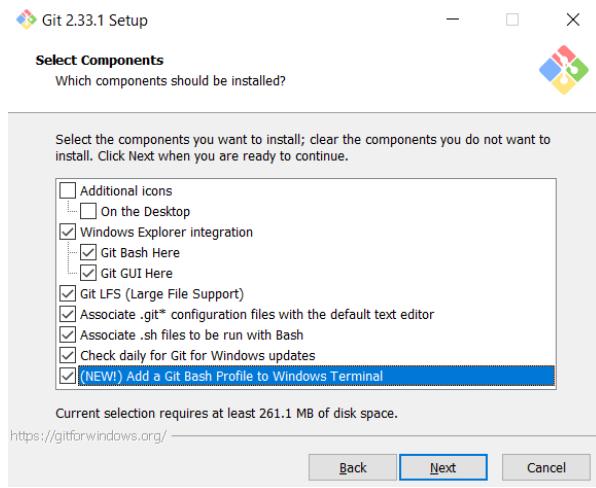


Imagen 4.42 Ventana de selección de componentes

La ventana siguiente es para seleccionar donde debería el programa de instalación colocar el acceso directo del programa; por default se deja así como está, en *Git*.

La ventana que sigue es para escoger que editor se quiere que Git use, por default se deja en *Use Vim (the ubiquitous text editor) as Git's default editor*.

La ventana a continuación, permite ajustar el nombre inicial a una rama en nuevos repositorios. Por default, la rama inicial cuando se crean nuevos repositorios es *master*, sin embargo, recomiendo seleccionar la segunda opción de la ventana de la Imagen 4.43, ya que como se va a hacer un repositorio remoto en GitHub, la rama principal en GitHub de todo nuevo repositorio es *main*, por lo que configurar el programa desde un principio a *main*, puede ahorrar cambiar el nombre en un futuro.

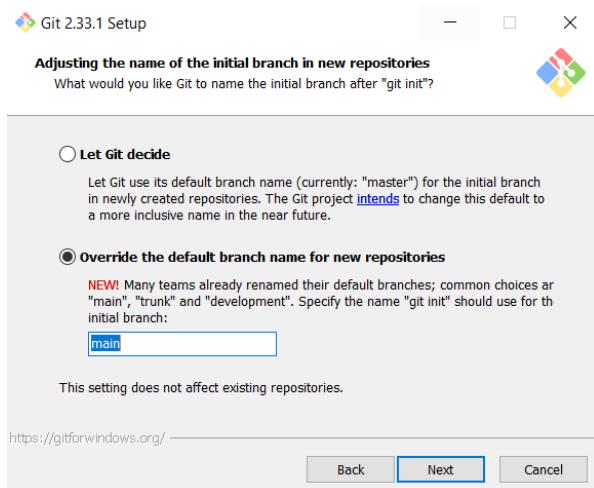


Imagen 4.43 Ventana de ajuste del nombre de la rama
inicial en un repositorio nuevo

La ventana que sigue es para ajustar el entorno PATH, más que nada para volverse más amigable en entorno de trabajo. En este caso se acepta la segunda opción y esto permite que se puede usar Git desde la línea de comandos, que es ya nativa en Windows; desde el software Git Bash, y entre otros.

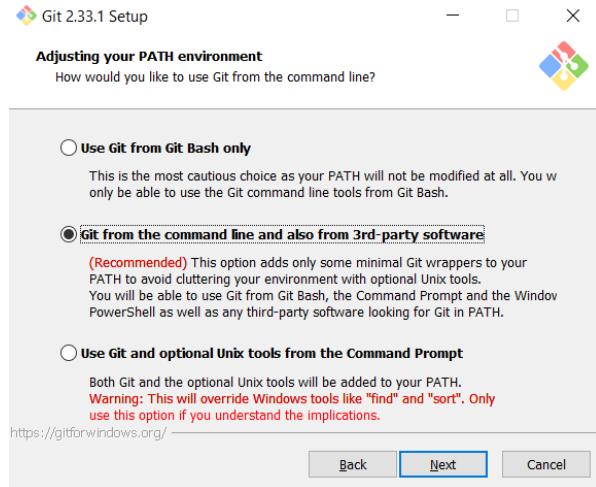


Imagen 4.44 Ventana de ajuste del entorno PATH

Las siguientes dos ventanas tienen que ver con la seguridad. La primera ventana es para elegir el programa cliente de Secure Shell que usará Git. Por default, se deja la primera opción *Use bundled OpenSSH*, que es un programa que viene ya incluido con Git. La segunda ventana es para escoger qué librería se usaría Git en conexiones HTTPS, igualmente por default, se deja la primera opción *Use the OpenSSL library* que es un sistema de seguridad que no viene en Windows, además que la gran mayoría del mundo del desarrollo de software la usa.

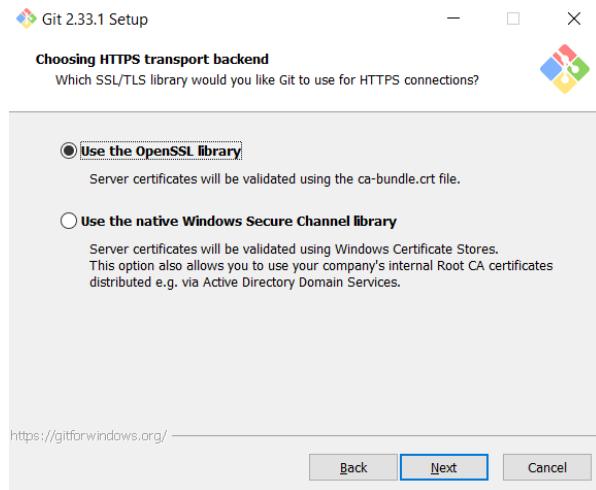


Imagen 4.45 Segunda ventana; ventana sobre la librería para conexiones seguras de HTTPS

La ventana posterior a las ventanas sobre la seguridad, solo pregunta que como se deben tratar a los finales de línea en los archivos de texto. Para evitar problemas ya que en el ámbito profesional puede haber desarrolladores con distintos sistemas operativos en sus equipos; se escoge la primera opción para que al principio los saltos de línea se reciban en Windows, al ser este mi entorno local de trabajo, pero cuando se envíen al repositorio se conviertan en Unix o Linux, brindando así una compatibilidad con todo el mundo.

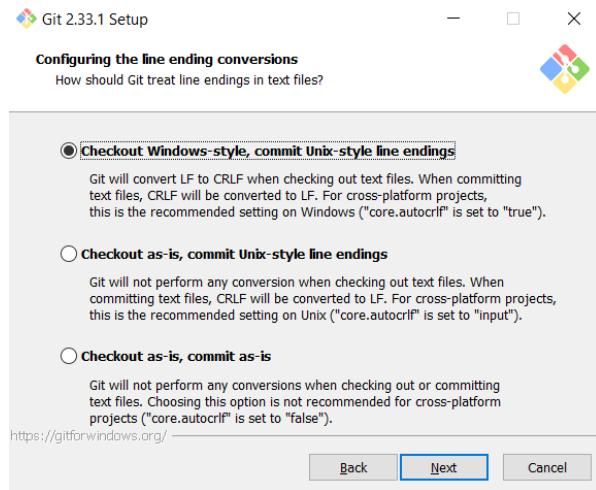


Imagen 4.46 Ventana sobre el tratamiento de los finales de línea

La ventana siguiente, da a elegir entre usar el emulador de Linux dentro de Windows o el sistema de Windows, cmd. Se recomienda usar el emulador, MinTTY, para acoplarse a la terminal ya que sus comandos son muy necesarios en varios ámbitos profesionales.

Luego, la ventana siguiente pregunta sobre cómo debe ser el comportamiento de *git pull*, por default se deja la primera opción ya que es el comportamiento estándar de este comando de git.

Finalmente se tienen ventanas para configurar opciones extra, como la configuración del asistente de credenciales, la habilitación del almacenamiento en caché del sistema de archivos o la habilitación de enlaces simbólicos. También se tiene una ventana en la que brinda añadir opciones experimentales nuevas, que recomiendo no añadir ya que no son necesarios.

Ahora solo queda esperar un par de minutos para el proceso de instalación.

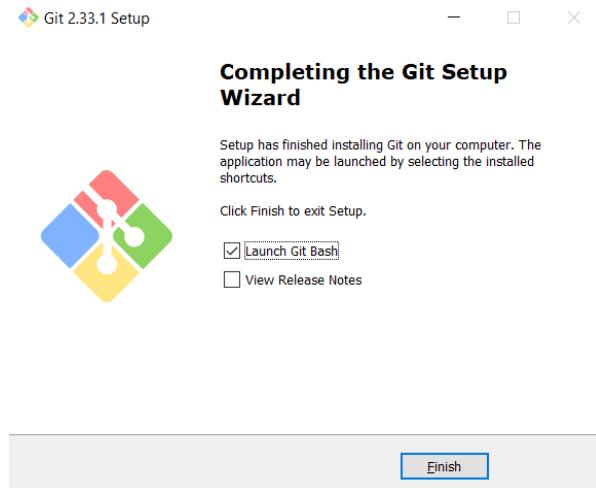


Imagen 4.47 Ventana de confirmación de la finalización de la instalación de Git

Por fin, se tiene ya instalado Git y Git Bash.

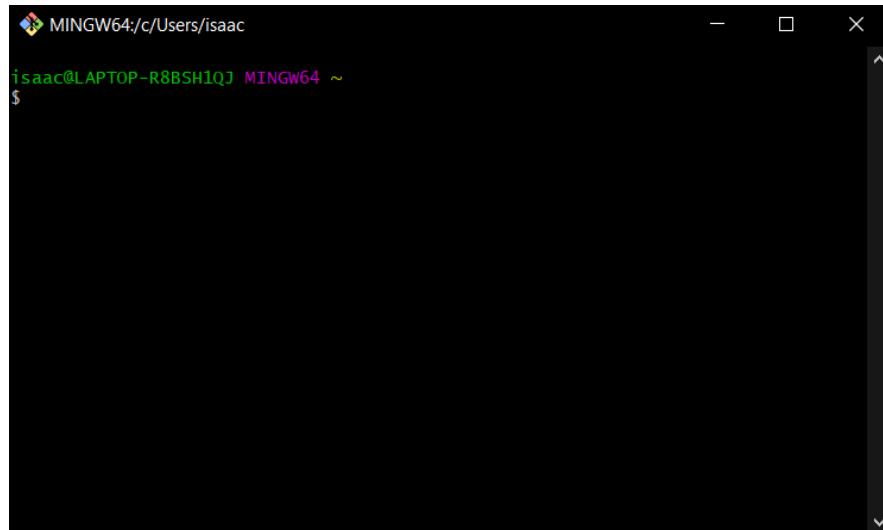


Imagen 4.48 Consola de Git Bash

4.4.2 Creación del Repositorio Local e Identificación

Para la utilización de Git Bash se necesita tener conocimientos de la terminal y la línea de comandos. Lo primero que se debe conocer es saber en dónde se está ubicado en la computadora, para ello se escribe *pwd* (print working directory) para que imprima el directorio actual.

Luego se escoge la ubicación en donde se desea tener el proyecto, en mi caso en la carpeta Documentos. Para acceder a esa carpeta se hace uso del comando *cd dir1* (change directory) para cambiar al directorio con nombre *dir1*; en mi caso el comando completo es *cd Documents/*. Si se desea regresar o cambiar al directorio anterior, se hace uso del comando *cd ..*.

Al estar en la ubicación deseada, ahora es momento de crear un directorio con el nombre del proyecto, el comando a utilizar es *mkdir dir1*, donde *dir1* es el nombre del proyecto. Para nombres con espacios, este nombre debe estar entre comillas (" ") porque si no la terminal llama a este directorio con la primera palabra escrita.

Al tener ya creado el directorio donde se tendrá almacenado el proyecto, solo se necesita acceder a él a través del comando, previamente visto, *cd dir1*.

Todos los pasos previamente mencionados se ven en la Imagen 4.49.

```

MINGW64:/c/Users/isaac/Documents/A Large Scale Fish Dataset
isaac@LAPTOP-R8BSH1QJ MINGW64 ~
$ pwd
/c/Users/isaac
isaac@LAPTOP-R8BSH1QJ MINGW64 ~
$ cd Documents/
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents
$ mkdir "A Large Scale Fish Dataset"
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents
$ cd "A Large Scale Fish Dataset"
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset
$ 
```

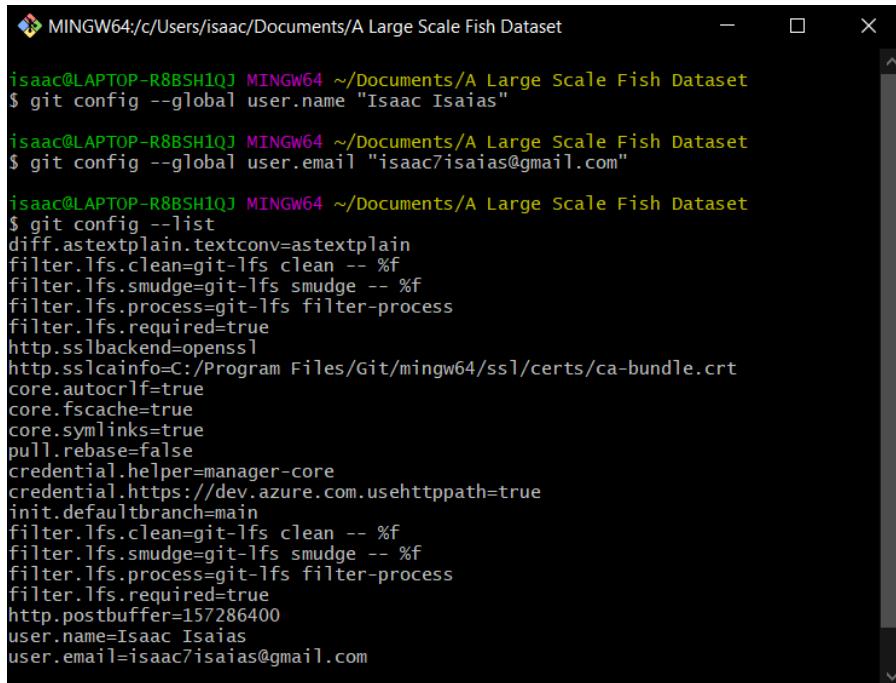
Imagen 4.49 Creación de un directorio a través de Git Bash

Ahora, antes de inicializar un repositorio local es necesario identificarse para que Git pueda saber quién está realizando los cambios en el proyecto. Los datos que se necesitan son un nombre y un correo electrónico.

Para agregar el nombre, se hace uso del comando `git config --global user.name "Nombre"`.

Para agregar el e-mail, se hace uso del comando `git config --global user.email "Email"`.

Para verificar que los datos hayan sido agregados en la configuración, se hace uso del comando `git config --list`.

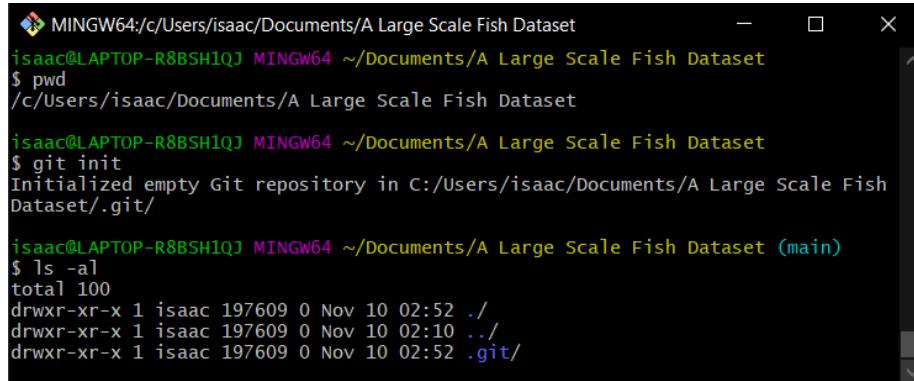


```
MINGW64:/c/Users/isaac/Documents/A Large Scale Fish Dataset
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset
$ git config --global user.name "Isaac Isaias"
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset
$ git config --global user.email "isaac7isaias@gmail.com"
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=true
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=main
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.postbuffer=157286400
user.name=Isaac Isaias
user.email=isaac7isaias@gmail.com
```

Imagen 4.50 Identificación del usuario con nombre y correo electrónico

Para crear un repositorio local, primero se ubica el directorio central en donde van a estar los archivos del proyecto y después de estar en el directorio central, solo se ejecuta el comando `git init`. Al crear el repositorio este crea un directorio oculto llamado `.git/` y ahí va a estar la base de datos de los cambios que se vayan a realizar.

También, solo basta con ver que entre paréntesis y de color azul se encuentra `main` que es el nombre de la rama principal del repositorio.



```

MINGW64:c/Users/isaac/Documents/A Large Scale Fish Dataset
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset
$ pwd
/c/Users/isaac/Documents/A Large Scale Fish Dataset

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset
$ git init
Initialized empty Git repository in C:/Users/isaac/Documents/A Large Scale Fish Dataset/.git/

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ ls -al
total 100
drwxr-xr-x 1 isaac 197609 0 Nov 10 02:52 .
drwxr-xr-x 1 isaac 197609 0 Nov 10 02:10 ../
drwxr-xr-x 1 isaac 197609 0 Nov 10 02:52 .git/

```

Imagen 4.51 Creación del repositorio local

El primer commit del repositorio local es añadir la licencia del dataset. Para ello, primero se mueve el archivo de la licencia al directorio central, después se comprueba el estatus del repositorio con el comando *git status* y git dice que aún no hay commits y que hay un archivo que no está trackeado o sin seguimiento. Estos archivos son aquellos que no se han tomado en cuenta para ser clonados, se puede decir que no son archivos “oficiales” del proyecto, por eso se debe hacer este procedimiento para que puedan ser parte del proyecto.

Después, para trackear el archivo, o darle seguimiento; se usa el comando *add file* para añadir al staging área, el área de memoria RAM donde se guardarán todos los cambios, estos archivos para posteriormente realizar el commit.

Algunos tips para escribir buenos commits son:

- Usar los verbos en imperativo y no en pasado, ya que así se expresará la acción que se realiza en el commit.
- No usar puntuaciones que no sean comas, ya sean puntos finales o puntos suspensivos.
- Ser corto y conciso, el commit debe tener como máximo 50 caracteres.
- Añadir un cuerpo de mensaje para añadir contexto al commit.

```

MINGW64:/c/Users/isaac/Documents/A Large Scale Fish Dataset
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    license.txt

nothing added to commit but untracked files present (use "git add" to track)

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git add license.txt

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   license.txt

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git commit -m "Initial commit"
[main (root-commit) 32f803c] Initial commit
  1 file changed, 384 insertions(+)
  create mode 100644 license.txt

```

Imagen 4.52 Procedimiento para realizar un commit

4.4.3 Creación del Repositorio Remoto en GitHub

Para poder alojar un repositorio en GitHub, lo primero que se debe tener es una cuenta en este sitio web. Después de iniciar sesión con el usuario de GitHub, se puede observar una barra de menú; en la parte derecha de la barra de menú se puede crear un repositorio. Para crearlo, se da clic en el símbolo más (+) y del menú de opciones se escoge *New repository*.

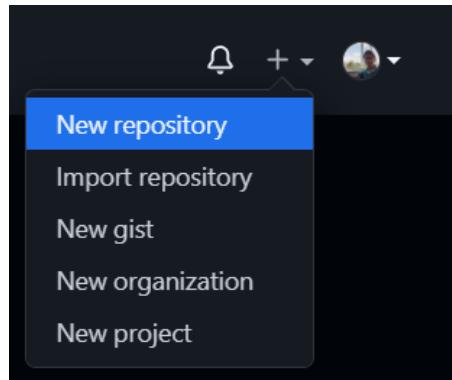


Imagen 4.53 Menú + de GitHub

Se llenan los datos correspondientes para la creación del repositorio remoto en GitHub. Después de llenarlos, se da clic en el botón *Create repository* para poder crearlo.

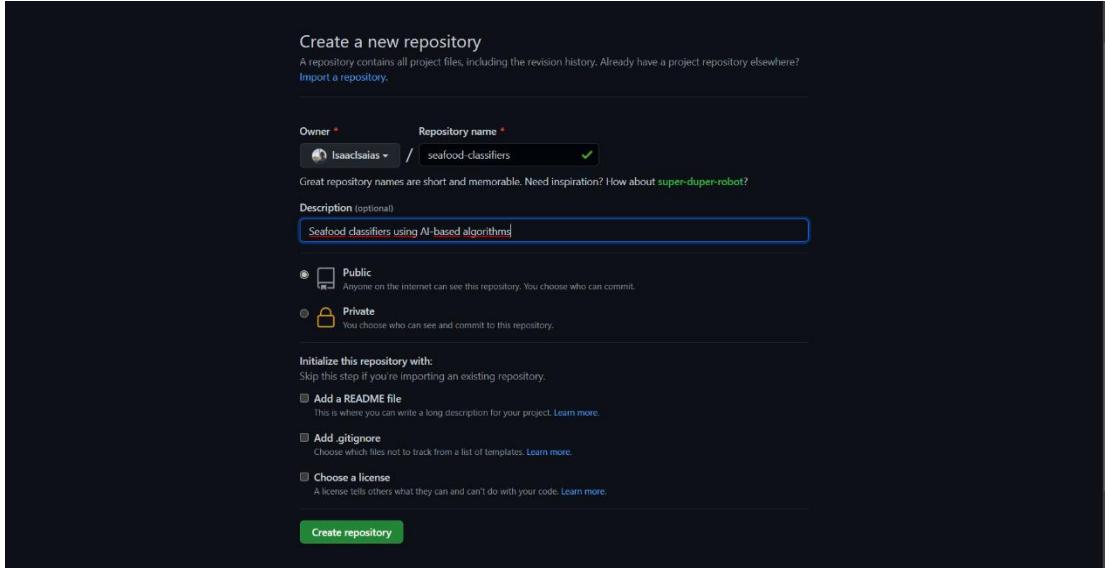


Imagen 4.54 Página para la creación de un nuevo repositorio en GitHub

En la siguiente página, ya aparece el repositorio creado solo que sin archivos. Como no se creó ni un archivo, esta página ofrece distintas opciones para poder crear un repositorio o realizar un *push* a un repositorio existente a través de la línea de comandos o importar código de otro repositorio.

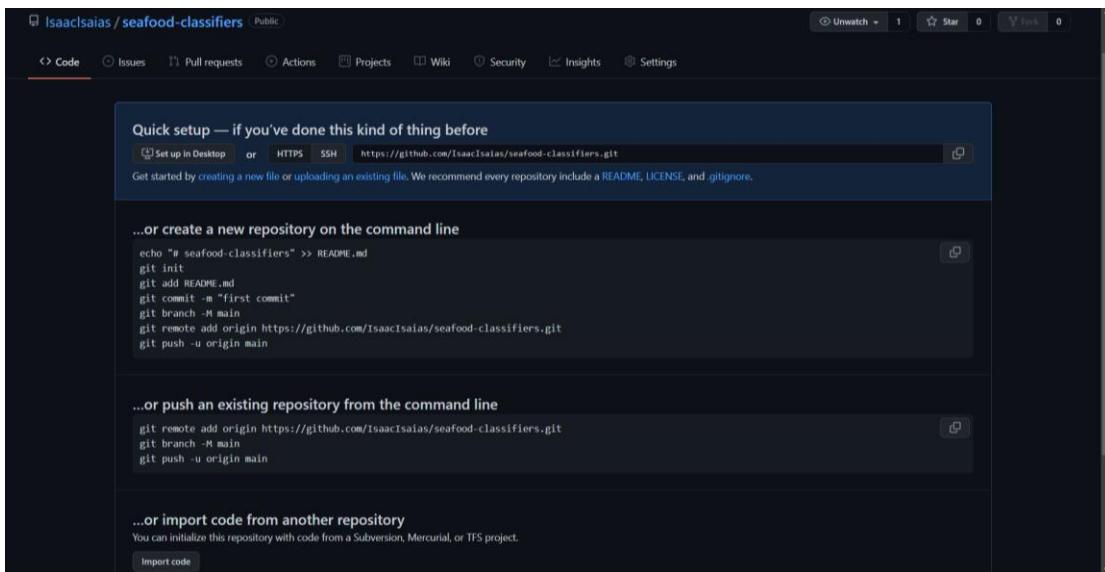


Imagen 4.55 Página del repositorio *seafood-classifiers*

El repositorio del proyecto se encuentra en el siguiente enlace:

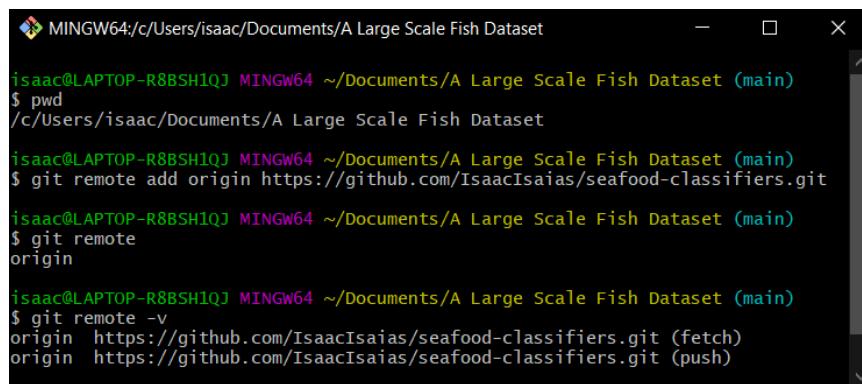
<https://github.com/IsaacIsaias/seafood-classifiers.git>

Para acceder a un repositorio a través de un navegador, lo primero que se tiene que hacer es escribir la URL de la página de inicio de GitHub (<https://github.com>); luego se escribe el nombre de usuario de GitHub que posea el repositorio (IsaacIsaias); después, se escribe el nombre del repositorio (seafood-classifiers); y por último se agrega el dominio .git.

4.4.4 Conexión entre los repositorios local y GitHub

Después de tener ya configurados tanto el repositorio local, como el remoto en GitHub; lo siguiente es hacer una vinculación entre ambos, con la finalidad de que los cambios que se hagan en el repositorio local también puedan ser reflejados en el remoto, y viceversa.

Para ello, primero se debe estar en el directorio central del proyecto y se ejecuta el comando `git remote add origin https://github.com/IsaacIsaias/seafood-classifiers.git`. Con este comando se añade un origen remoto de los archivos. Con `git remote` y `git remote -v` se comprueba de que se ha almacenado correctamente la URL; `git remote` muestra que hay un origen remoto llamado `origin`, mientras que `git remote -v` muestra que se tiene un `origin` para hacer `fetch` (para traer archivos) y un `origin` para hacer `push` (para enviar archivos).



```

MINGW64:/c/Users/isaac/Documents/A Large Scale Fish Dataset
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ pwd
/c/Users/isaac/Documents/A Large Scale Fish Dataset

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git remote add origin https://github.com/IsaacIsaias/seafood-classifiers.git

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git remote
origin

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git remote -v
origin https://github.com/IsaacIsaias/seafood-classifiers.git (fetch)
origin https://github.com/IsaacIsaias/seafood-classifiers.git (push)

```

Imagen 4.56 Conexión entre repositorio local/remoto

El comando `git push origin main` guarda los cambios del repositorio local en GitHub, sin embargo, el comando que se va a utilizar es `git push -u origin main`. Los dos comandos funcionan de la misma manera, la única diferencia que hace la `-u` extra, es que con el segundo comando de `push`, ya no es necesario ocupar argumentos con el `git pull`.

Unos segundos después de ejecutar el comando, se pedirá que se inicie sesión con el usuario de GitHub.

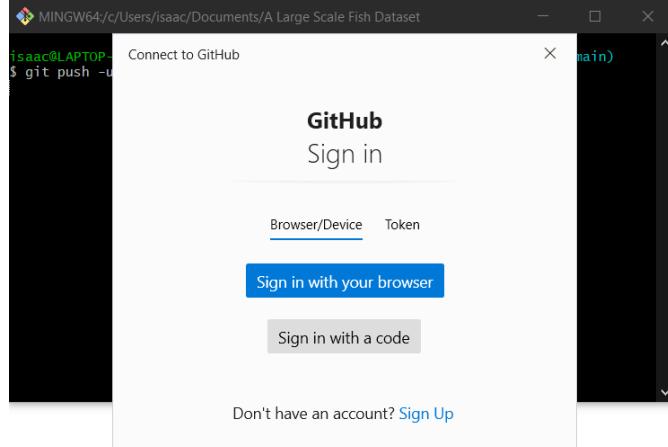


Imagen 4.57 Petición de inicio de sesión

En mi caso, inicio sesión desde el navegador y este me redirecciona a una página donde me solicita autorizar el uso del helper Git Credential Manager. Este nuevo helper proporciona un almacenamiento seguro de las credenciales Git para Windows.

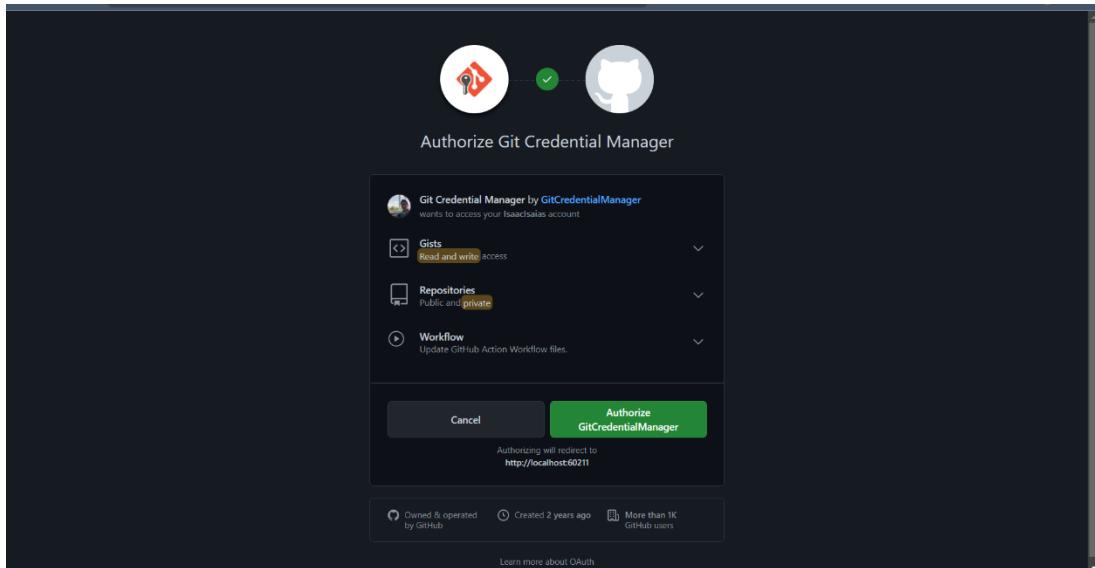


Imagen 4.58 Petición de autorización del GCM

Tras autorizar el helper, se acepta el *push* hacia GitHub y se comienzan a enviar los archivos.

```

MINGW64:/c/Users/isaac/Documents/A Large Scale Fish Dataset
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 5.92 KiB | 1.97 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/IsaacIsaias/seafood-classifiers.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

```

Imagen 4.59 Confirmación del *push* del primer commit

Al entrar nuevamente a la página del repositorio remoto en GitHub, se observa que ya existe un commit y se encuentra ya reflejado la licencia del dataset.

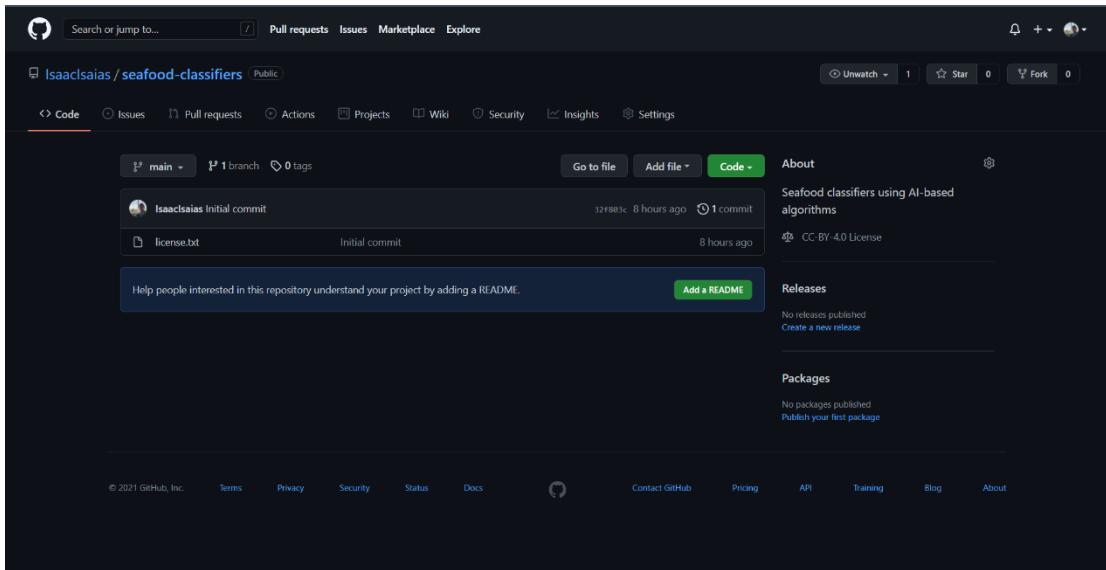


Imagen 4.60 Página del repositorio *seafood-classifiers* con el primer commit

Como último paso, queda realizar los commits necesarios para poder tener todos los archivos del proyecto disponibles en GitHub.

Cuando se tratan de subir un conjunto de archivos de tamaño considerable, se debe ejecutar el siguiente comando: *git config --global http.postBuffer 524288000*.

4.5 CNN para un dataset de mariscos

Para la implementación de una Red Neuronal Convolucional, se va a trabajar con dos entornos de trabajo distintos. El primer entorno es con los Jupyter Notebooks de Kaggle y el segundo es con los Notebooks de Colab. Cada entorno posee sus ventajas y desventajas, por tales razones se ejecuta el código en ambos entornos.

Una de las principales ventajas de realizar el código en Kaggle, es que no es necesario clonar algún repositorio para obtener datasets, debido a que la misma plataforma los brinda. En cambio, con Colab es necesario tener en el Google Drive un dataset creado por uno mismo o un dataset de un repositorio de GitHub, añadiendo también que puede que consuma mucho tiempo la clonación de este.

Una desventaja en Kaggle es que suelen haber errores con las librerías, un ejemplo que me pasó es que no reconocía a Keras, por lo que tenía que llamarlo de forma completa a través de TensorFlow, ocasionando que el código tenga más caracteres.

Ambos, tanto Kaggle y Colab, permiten tener varias horas gratuitas de GPUs o TPUs, pero la diferencia es que Kaggle ofrece 42 horas gratis a la semana y estás 42 horas, se pueden aprovechar de forma seguida sin interrupción alguna; sin embargo, Colab no permite tanto tiempo seguido estar usando estos servicios de cómputo, por lo que se te pueden desconectar aun así se esté entrenando una red neuronal u otras aplicaciones.

La principal ventaja de Colab es su entorno de trabajo porque es muy intuitivo y práctico, así como la gran diversa cantidad de funcionalidades que se pueden hacer en la plataforma.

Después de ver algunas ventajas y desventajas, se prosigue al desarrollo del modelo.

Prácticamente, el código es el mismo en ambas plataformas, solo que hay ligeros cambios. En el código de Colab se clona un repositorio; mientras que en Kaggle no, sino se selecciona un dataset. El otro cambio entre códigos, es la forma de importar las librerías y la forma de obtener las rutas de los archivos y su forma de representación en el dataframe.

Como se ha visto a lo largo de este reporte, la API a ocupar es Keras y el backend es TensorFlow, herramientas necesarias para desarrollar el código a través de Python 3.

4.5.1 Clonación del Repositorio de GitHub en Colab

Para clonar el repositorio previamente realizado con el dataset de Kaggle, se ejecuta el comando `!git clone https://github.com/IsaacIsaias/seafood-classifiers.git`.

```
[ ] 1 !git clone https://github.com/IsaacIsaias/seafood-classifiers.git
Cloning into 'seafood-classifiers'...
remote: Enumerating objects: 18501, done.
remote: Total 18501 (delta 0), reused 0 (delta 0), pack-reused 18501
Receiving objects: 100% (18501/18501), 3.23 GiB | 37.24 MiB/s, done.
Resolving deltas: 100% (17/17), done.
Checking out files: 100% (18432/18432), done.
```

Imagen 4.61 Clonación del Repositorio de GitHub en Colab

4.5.2 Importación de Librerías

Posteriormente se importan las librerías y se renombran si es que lo necesitan, por convención de la comunidad de Inteligencia Artificial y Ciencia de Datos.

```
[ ] 1 import warnings
2 warnings.filterwarnings('ignore')
3
4 import numpy as np
5 import pandas as pd
6
7 from pathlib import Path
8 import os.path
9 import glob
10
11 import matplotlib.pyplot as plt
12 import seaborn as sns
13
14 from sklearn.model_selection import train_test_split
15 from sklearn.metrics import confusion_matrix, classification_report
16
17 import tensorflow as tf
18 from keras.preprocessing.image import ImageDataGenerator
19 from keras import Input, Model
20 from keras.layers import Dense, Conv2D, GlobalAveragePooling2D, MaxPooling2D
21 from keras.callbacks import EarlyStopping
```

Imagen 4.62 Importación de Librerías en Colab

4.5.3 Manipulación de Datos

En esta sección se trata sobre toda la obtención, el manejo y la transformación de datos

4.5.3.1 Obteniendo Datos

Con el comando `os.listdir(directorio)` se obtiene un directorio en forma de lista.

Para la obtención de las rutas de los archivos, primero se crea una lista vacía para que esta sea la ubicación donde se van a almacenar. El ciclo `for` es usado de tal forma, que la función `iglob` del módulo `glob`, encuentre a todos los archivos `.png` de la carpeta `Fish_Dataset`, ya que las todas las imágenes del dataset tienen ese formato; y almacene su ruta de archivo en una string para posteriormente la ruta sea añadida a la lista vacía.

Para la obtención de las etiquetas es algo similar, se almacena en forma de lista haciendo uso de la función `list()`, y como argumento de esta función se tiene un mapeo de una función lambda. La función lambda consiste en ir separando la ruta de archivo por secciones y en forma de strings, hasta que solamente se quede la etiqueta a la que hace referencia la ruta del archivo.

```

Data Manipulation

Getting Data

[ ] 1 os.listdir('/content/seafood-classifiers/Fish_Dataset/')

['Sea Bass',
 'Black Sea Sprat',
 'Hourse Mackerel',
 'Red Mullet',
 'Trout',
 'Striped Red Mullet',
 'Shrimp',
 'Red Sea Bream',
 'Gilt-Head Bream']

[ ] 1 # Getting filepaths
2 filepaths = []
3 for filepath in glob.iglob('/content/seafood-classifiers/Fish_Dataset/**/*.*.png', recursive=True):
4     filepaths.append(filepath)
5
6 # Getting labels
7 labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))

```

Imagen 4.63 Obtención de las rutas de archivos y etiquetas con Colab

A continuación se muestran los primeros 10 ejemplos de la obtención de rutas de archivos y su respectiva etiqueta o clase.

```
[ ] 1 filepaths
['/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00232.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00688.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00464.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00963.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00603.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00984.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00182.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/01000.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00071.png',
 '/content/seafood-classifiers/Fish_Dataset/Sea Bass/Sea Bass/00505.png',
```

Imagen 4.64 Primeros diez ejemplos de la obtención de rutas de archivos

```
[ ] 1 labels
['Sea Bass',
 'Sea Bass',
```

Imagen 4.65 Primeros diez ejemplos

de la obtención de etiquetas

En las siguientes líneas de código, se aprecia como se fue construyendo la línea de código para obtener una etiqueta con el uso de *os.path*, para que después se pudiera generalizar este proceso en la función lambda.

```
[ ] 1 os.path.split('/content/seafood-classifiers/Fish_Dataset/Black Sea Sprat/Black Sea Sprat/00001.png')
('/content/seafood-classifiers/Fish_Dataset/Black Sea Sprat/Black Sea Sprat',
 '00001.png')

[ ] 1 os.path.split('/content/seafood-classifiers/Fish_Dataset/Black Sea Sprat/Black Sea Sprat/00001.png')[0]
'/content/seafood-classifiers/Fish_Dataset/Black Sea Sprat/Black Sea Sprat'

[ ] 1 os.path.split(os.path.split('/content/seafood-classifiers/Fish_Dataset/Black Sea Sprat/Black Sea Sprat/00001.png')[0])
('/content/seafood-classifiers/Fish_Dataset/Black Sea Sprat',
 'Black Sea Sprat')

[ ] 1 os.path.split(os.path.split('/content/seafood-classifiers/Fish_Dataset/Black Sea Sprat/Black Sea Sprat/00001.png')[0])[1]
'Black Sea Sprat'
```

Imagen 4.66 Explicación del surgimiento de la generalización de la función lambda

4.5.3.2 Generación de Series

Al ya tener las rutas de archivos y las etiquetas, se busca convertirlas en Series de Pandas para que de los elementos pueda tener su índice.

```
[ ] 1 # Filepaths/Labels Series generation
2 filepaths = pd.Series(filepaths, name='Filepath')
3 labels = pd.Series(labels, name='Label')
```

Imagen 4.67 Generación de Series con Pandas con Colab

Aquí se muestra como cada uno de los 18,000 elementos, tanto de las rutas de los archivos como de las etiquetas; ya tiene su índice y manteniendo su misma posición en ambas columnas.

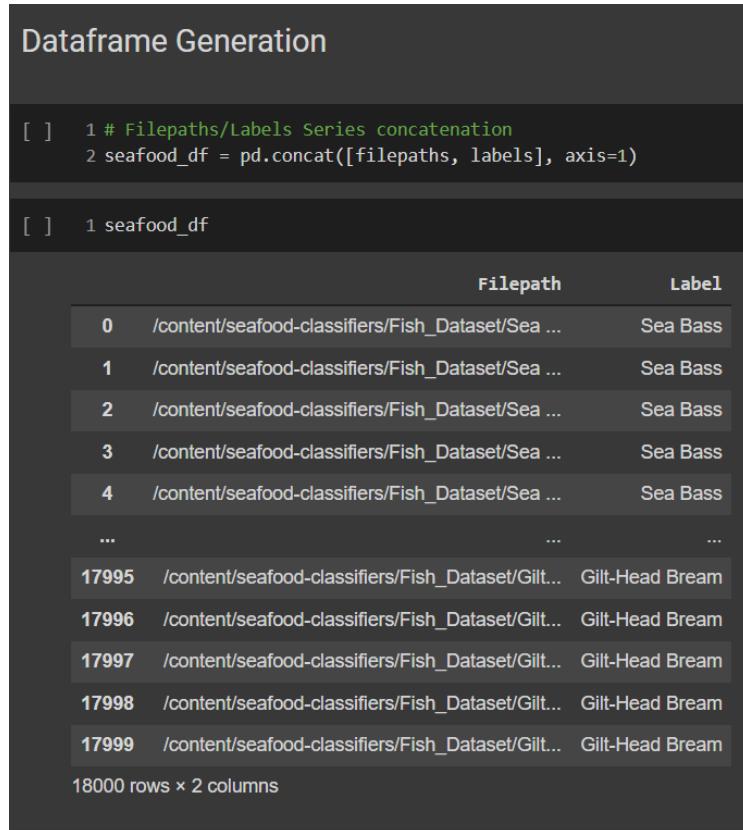
```
[ ] 1 filepaths
0      /content/seafood-classifiers/Fish_Dataset/Sea ...
1      /content/seafood-classifiers/Fish_Dataset/Sea ...
2      /content/seafood-classifiers/Fish_Dataset/Sea ...
3      /content/seafood-classifiers/Fish_Dataset/Sea ...
4      /content/seafood-classifiers/Fish_Dataset/Sea ...
...
17995   /content/seafood-classifiers/Fish_Dataset/Gilt...
17996   /content/seafood-classifiers/Fish_Dataset/Gilt...
17997   /content/seafood-classifiers/Fish_Dataset/Gilt...
17998   /content/seafood-classifiers/Fish_Dataset/Gilt...
17999   /content/seafood-classifiers/Fish Dataset/Gilt...
Name: Filepath, Length: 18000, dtype: object

[ ] 1 labels
0          Sea Bass
1          Sea Bass
2          Sea Bass
3          Sea Bass
4          Sea Bass
...
17995    Gilt-Head Bream
17996    Gilt-Head Bream
17997    Gilt-Head Bream
17998    Gilt-Head Bream
17999    Gilt-Head Bream
Name: Label, Length: 18000, dtype: object
```

Imagen 4.68 Series de las rutas y las etiquetas

4.5.3.3 Generación del DataFrame

Para generar un DataFrame y se pueda tener una tabla relacional entre ambas Series, o columnas; lo que se hace es aplicar la función `concat()` de Pandas para que las dos Series se unan. El `axis=1` es para señalar que se deben unir a través del eje vertical, es decir, a través de las columnas.



The screenshot shows a Jupyter Notebook cell titled "Dataframe Generation". The code cell contains the following Python code:

```
[ ] 1 # Filepaths/Labels Series concatenation
2 seafood_df = pd.concat([filepaths, labels], axis=1)

[ ] 1 seafood_df
```

Below the code cell is a table representation of the DataFrame. The table has two columns: "Filepath" and "Label". The first five rows show entries for "Sea Bass". Rows 17995 through 17999 show entries for "Gilt-Head Bream". The final row indicates there are 18000 rows by 2 columns.

	Filepath	Label
0	/content/seafood-classifiers/Fish_Dataset/Sea ...	Sea Bass
1	/content/seafood-classifiers/Fish_Dataset/Sea ...	Sea Bass
2	/content/seafood-classifiers/Fish_Dataset/Sea ...	Sea Bass
3	/content/seafood-classifiers/Fish_Dataset/Sea ...	Sea Bass
4	/content/seafood-classifiers/Fish_Dataset/Sea ...	Sea Bass
...
17995	/content/seafood-classifiers/Fish_Dataset/Gilt... Gilt-Head Bream	Gilt-Head Bream
17996	/content/seafood-classifiers/Fish_Dataset/Gilt... Gilt-Head Bream	Gilt-Head Bream
17997	/content/seafood-classifiers/Fish_Dataset/Gilt... Gilt-Head Bream	Gilt-Head Bream
17998	/content/seafood-classifiers/Fish_Dataset/Gilt... Gilt-Head Bream	Gilt-Head Bream
17999	/content/seafood-classifiers/Fish_Dataset/Gilt... Gilt-Head Bream	Gilt-Head Bream
18000	rows × 2 columns	

Imagen 4.69 Generación del DataFrame con Pandas

Luego, se busca botar o soltar de alguna forma las imágenes que no se van a requerir para ser entrenadas en el modelo. Estas son imágenes Ground Truth que previamente fueron elaboradas por el equipo que realizó el dataset y cuyas etiquetas terminan con GT.

La forma de hacerlo es aplicando una función lambda en la que si las últimas dos letras de una etiqueta dicen GT, que estas se conviertan en objetos nulos o indefinidos. Es por esta razón que se convierte a valores de tipo NaN (Not a Number). Posteriormente con la función

`dropna()` se sueltan dichos valores del DataFrame. Los demás elementos que se añadieron son para seguir manteniendo el 100% de los datos, se revuelvan y se resetee el índice.

Para verificar que únicamente quedan 9,000 imágenes, se aplica el método `info()` al DataFrame para ver cuantas entradas no nulas se tienen por cada columna y el obtener información sobre el índice.

```
[ ] 1 # Drop Ground Truth images
2 seafood_df['Label'] = seafood_df['Label'].apply(lambda x: np.NaN if x[-2:] == 'GT' else x)
3 seafood_df = seafood_df.dropna(axis=0).sample(frac=1.0, random_state=27).reset_index(drop=True)

[ ] 1 seafood_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9000 entries, 0 to 8999
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   Filepath  9000 non-null   object  
 1   Label     9000 non-null   object  
dtypes: object(2)
memory usage: 140.8+ KB
```

Imagen 4.70 Función para soltar las imágenes GT y su comprobación

Ahora, para poder saber cuántos elementos se tiene de cada clase se usa el método `value_counts()` en el DataFrame, usando como referencia al *Label*. Para obtener el nombre de cada uno de las clases de *Label*, se usa el método `unique()`.

```
[ ] 1 seafood_df['Label'].value_counts()

Shrimp          1000
Trout           1000
Sea Bass        1000
Striped Red Mullet  1000
Black Sea Sprat 1000
Red Sea Bream   1000
Red Mullet      1000
Gilt-Head Bream 1000
Hourse Mackerel 1000
Name: Label, dtype: int64

[ ] 1 seafood_df['Label'].unique()

array(['Trout', 'Shrimp', 'Striped Red Mullet', 'Red Sea Bream',
       'Gilt-Head Bream', 'Sea Bass', 'Black Sea Sprat', 'Red Mullet',
       'Hourse Mackerel'], dtype=object)
```

Imagen 4.71 Métodos de conteo

4.5.3.4 Mostrando Datos

Haciendo uso de la API orientada a objetos, se muestra una imagen de cada una de las clases del dataset, esto más que nada para poder tener un mejor control sobre el gráfico. El gráfico

muestra tres filas y tres columnas para que se representen los 9 tipos de mariscos distintos, junto con sus dimensiones, que por default son de (445, 590, 3); y su respectiva etiqueta.

Showing Data

```
[ ]  1 fig, ax = plt.subplots(nrows=3, ncols=3, figsize=(13,11), constrained_layout=True)
  2 ax = ax.flatten()
  3 j = 0
  4
  5 for i in seafood_df['Label'].unique():
  6     ax[j].imshow(plt.imread(seafood_df[seafood_df['Label'] == i].iloc[0,0]))
  7     ax[j].set_title(i)
  8     j = j + 1
```

Imagen 4.72 Código para mostrar un elemento de cada clase

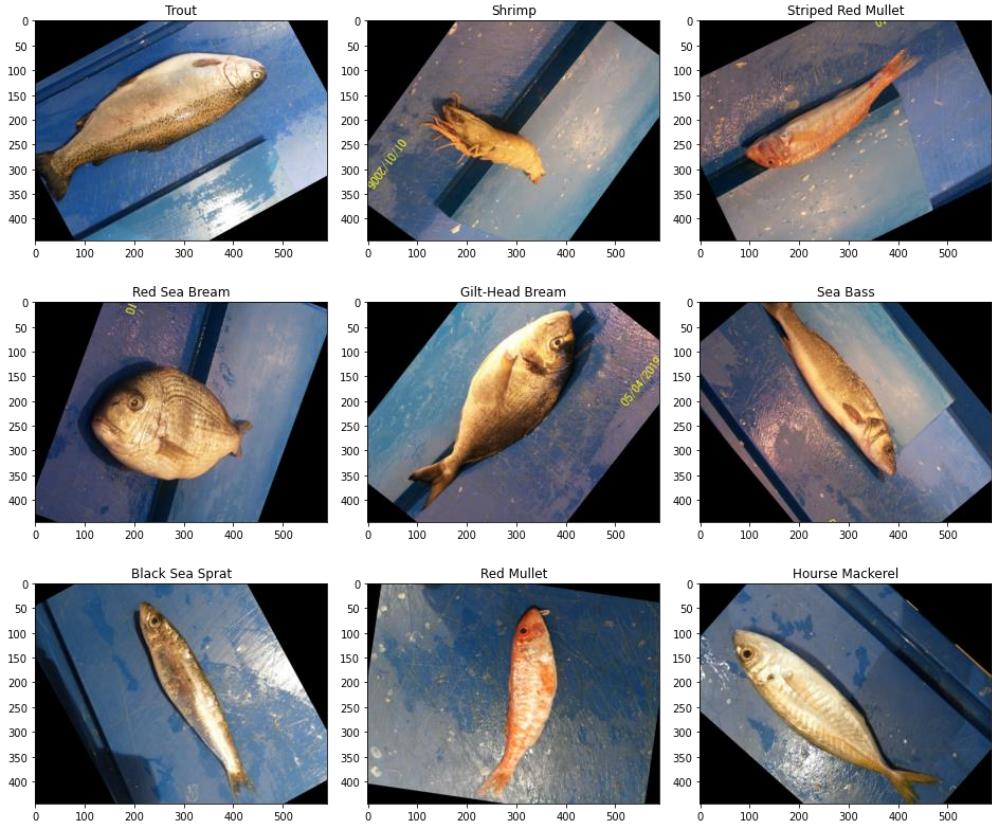


Imagen 4.73 Gráfico mostrando cada uno de los mariscos distintos del dataset

4.5.3.5 Contando los Datos

Otra manera de contar los datos de un dataset es haciendo uso de los gráficos. Con seaborn se pueden realizar gráficos de una manera más rápida y sencilla que con matplotlib.pyplot, además que estos gráficos buscan ser más llamativos o mejores.

```
Counting Data

[ ]  1 fig = plt.figure(figsize=(20,10))
2 sns.countplot(x = seafood_df['Label'])
3
4 # FutureWarning: Pass the following variable as a keyword arg: x.
5 # From version 0.12, the only valid positional argument will be `data`,
6 # and passing other arguments without an explicit keyword will result in
7 # an error or misinterpretation.

<matplotlib.axes._subplots.AxesSubplot at 0x7fd9b72f5d90>
```

Imagen 4.74 Código para visualizar la cantidad de datos en un gráfico con seaborn

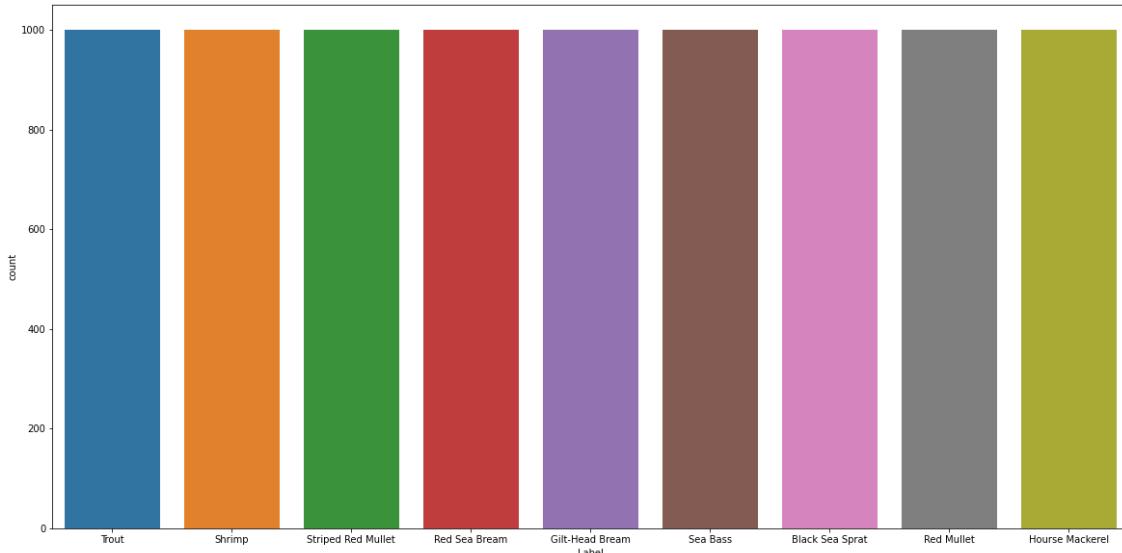


Imagen 4.75 Gráfico de seaborn con el total de elementos de cada una de las clases

4.5.4 Implementación del Modelo

Después de haber obtenidos los datos desde un repositorio y se hayan manipulado los datos del dataset de manera adecuada, es momento de la implementación de una CNN.

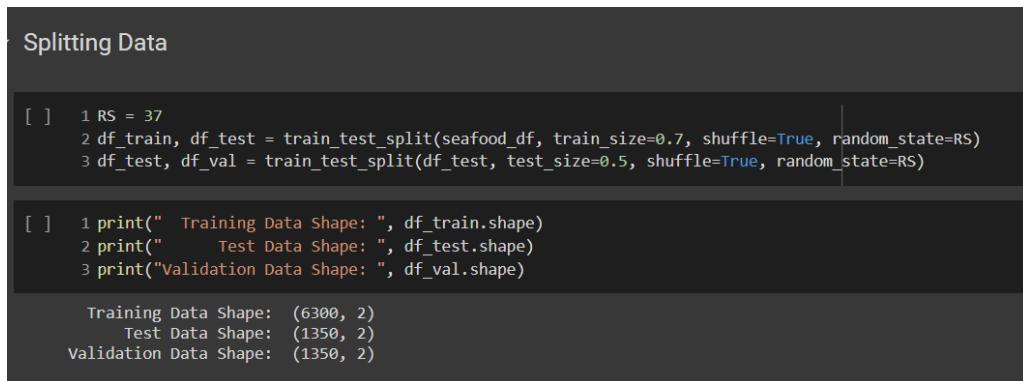
4.5.4.1 División de los Datos

En esta sección, se separan los datos en tres conjuntos. Los conjuntos son el de entrenamiento, el de test y el de validación. Para ello, se ocupa el módulo de *sklearn*, *train_test_split*. La primera línea es una constante y esta constante sirve para que cualquier persona pueda replicar exactamente la distribución de los datos que se tiene en este modelo.

La segunda línea es para asignarle al conjunto de entrenamiento (*df_train*) el 70% de los datos totales y que el conjunto de comprobación (*df_test*) tenga el 30% restante. Sin embargo, en la tercera línea pasa algo similar, solo que se reparten ese 30% restante en partes iguales a los dos conjuntos pequeños, al de comprobación y al de validación (*df_val*). Simplemente se podía haber hecho solo la primera y segunda línea para el entrenamiento y la comprobación, pero como buena práctica es mejor separarlos como se hizo, en tres conjuntos.

El set de entrenamiento (*df_train*) es para que el algoritmo pueda generalizar y crear información; con el set de validación (*df_val*) se puede hacer la prueba de que tan bien estuvo el modelo, ya que brindan la oportunidad de cambiar hiperparámetros para que el algoritmo sea preciso y el adecuado. Finalmente, el set de comprobación (*df_test*) se usa para evaluar el comportamiento del modelo. Todo esto es, como en toda área de conocimiento, por cuestiones de ética. Todo algoritmo y modelo solo debe ser probado con datos jamás antes vistos, con la finalidad de obtener resultados más certeros con la realidad.

Por último, las últimas tres líneas del código son para imprimir el tamaño de los tres sets.



```
[ ] 1 RS = 37
2 df_train, df_test = train_test_split(seafood_df, train_size=0.7, shuffle=True, random_state=RS)
3 df_test, df_val = train_test_split(df_test, test_size=0.5, shuffle=True, random_state=RS)

[ ] 1 print(" Training Data Shape: ", df_train.shape)
2 print(" Test Data Shape: ", df_test.shape)
3 print("Validation Data Shape: ", df_val.shape)

Training Data Shape: (6300, 2)
Test Data Shape: (1350, 2)
Validation Data Shape: (1350, 2)
```

Imagen 4.76 División de los datos en tres sets

Los datos de los conjuntos quedaron distribuidos de la siguiente manera por categoría:

Counting Data	
[] 1 df_train['Label'].value_counts()	
Shrimp	715
Trout	708
Hourse Mackerel	705
Sea Bass	701
Gilt-Head Bream	701
Black Sea Sprat	695
Striped Red Mullet	694
Red Mullet	694
Red Sea Bream	687
Name: Label, dtype: int64	
[] 1 df_test['Label'].value_counts()	
Red Mullet	162
Hourse Mackerel	156
Striped Red Mullet	156
Sea Bass	155
Black Sea Sprat	154
Gilt-Head Bream	148
Shrimp	147
Red Sea Bream	139
Trout	133
Name: Label, dtype: int64	
[] 1 df_val['Label'].value_counts()	
Red Sea Bream	174
Trout	159
Gilt-Head Bream	151
Black Sea Sprat	151
Striped Red Mullet	150
Sea Bass	144
Red Mullet	144
Hourse Mackerel	139
Shrimp	138
Name: Label, dtype: int64	

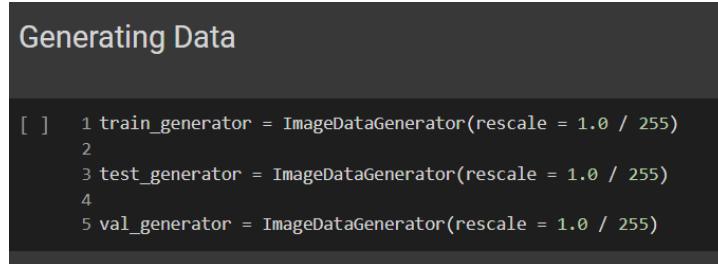
Imagen 4.77 Distribución de los datos entre los sets

4.5.4.2 Generación de Datos

En esta parte del código se va a trabajar con *ImageDataGenerator*. Esta herramienta permite generar lotes de datos de imágenes de tensores y *data augmentation*, o aumento de datos, mientras se entrenando el modelo. Esta es una herramienta especial cuando no se tienen los datos suficientes para realizar un buen modelo de aprendizaje.

Para este proyecto, el dataset cuenta con una considerable cantidad de datos, debido a que los creadores de este, previamente hicieron un *data augmentation* al pequeño conjunto de datos que lograron recabar.

Como no es necesario aumentar los datos, solo se realiza un ajuste a las imágenes, el cual es un reescalado para que los valores de los píxeles de las imágenes estén entre 0 y 1.



```
[ ] 1 train_generator = ImageDataGenerator(rescale = 1.0 / 255)
2
3 test_generator = ImageDataGenerator(rescale = 1.0 / 255)
4
5 val_generator = ImageDataGenerator(rescale = 1.0 / 255)
```

Imagen 4.78 Reescalado de las imágenes

Después de haber hecho un reescalado a las imágenes, se hace uso del método *flow_from_dataframe()*. Este método toma al DataFrame de Pandas y la ruta a un directorio, y luego genera lotes de datos aumentados o normalizados; esto ayuda a dejar más espacio en la memoria cuando se están leyendo o preprocesando imágenes al mismo tiempo.

A continuación se muestra el código empleado para el método *flow_from_dataframe*. Primero se estableció que tamaño de las imágenes durante el entrenamiento es de (256, 256) píxeles. También se establecieron otras dos constantes, BATCH_SIZE es la cantidad de imágenes por lote y SEED es la semilla o la constante a definir para que otras personas puedan replicarlo.

Los parámetros que pide el método *flow_from_dataframe* para los tres sets, son valores del DataFrame hecho con Pandas, como lo son la Serie Filepath para la columna x y la Serie Label para la columna y. El parámetro *dataframe* es para seleccionar de donde va a tomar los datos, pero para esto, cada variable tiene sus propios sets previamente establecidos cuando se reescalaban las imágenes. Por último, hay parámetros relacionados con el tipo de imágenes y clasificación que se va a realizar; *color_mode* es para denotar cuantos canales tienen las imágenes, al ser imágenes RGB, se escribe como argumento ‘rgb’; el *class_mode* es para

denotar que tipo de clase es la salida, en este caso al tener más de dos salidas la clasificación, se escribe como argumento ‘categorical’, pues es una clasificación multiclas.

```
[ ] 1 TARGET_SIZE=(256,256)
2 BATCH_SIZE=30
3 SEED=17
4
5 train_set = train_generator.flow_from_dataframe(
6     dataframe=df_train,
7     x_col='Filepath',
8     y_col='Label',
9     target_size=TARGET_SIZE,
10    color_mode='rgb',
11    class_mode='categorical',
12    batch_size=BATCH_SIZE,
13    shuffle=True,
14    seed=SEED,
15 )
16
17 test_set = test_generator.flow_from_dataframe(
18     dataframe=df_test,
19     x_col='Filepath',
20     y_col='Label',
21     target_size=TARGET_SIZE,
22     color_mode='rgb',
23     class_mode='categorical',
24     batch_size=BATCH_SIZE,
25     shuffle=False,
26 )
27
28 val_set = val_generator.flow_from_dataframe(
29     dataframe=df_val,
30     x_col='Filepath',
31     y_col='Label',
32     target_size=(224, 224),
33     color_mode='rgb',
34     class_mode='categorical',
35     batch_size=BATCH_SIZE,
36     shuffle=True,
37     seed=SEED,
38 )

Found 6300 validated image filenames belonging to 9 classes.
Found 1350 validated image filenames belonging to 9 classes.
Found 1350 validated image filenames belonging to 9 classes.
```

Imagen 4.79 Código empleado para el módulo *flow_from_dataframe*

4.5.5 Desarrollo del Modelo

Existen dos formas de definir una estructura de red neuronal, estos son como un modelo Secuencial o usando la API funcional.

Un modelo Secuencial es útil para definir de una manera rápida una pila lineal de capas, es decir, donde una capa sigue directamente a la capa anterior sin ninguna ramificación.

Para poder construir modelos con topología no lineal, con capas compartidas, o con múltiples entradas o salidas; se hace uso de la API funcional, ya que es mucho más flexible pues brinda la total libertad sobre el diseño de la red neuronal profunda.

4.5.5.1 Arquitectura del Modelo

El modelo de este proyecto va a emplear la API funcional como estructura de la CNN. Lo primero que se necesitar es crear un nodo de entrada para la red, para eso se usa la clase *Input* y esta clase usa como argumento el tamaño de entrada de las imágenes.

El primer tipo de capa es la capa de convolución *Conv2D*, capa especializada para las convoluciones espaciales sobre las imágenes. Esta capa crea un filtro de convolución que se convoluciona con la entrada de la capa para producir un tensor de salida. El primer parámetro que se especifica son la cantidad de filtros, el segundo parámetro es el tamaño del filtro, y el tercer parámetro es la función de activación.

El segundo tipo de capa es la capa de agrupación *MaxPooling2D*, esta capa se encarga de realizar la operación de agrupación máxima para datos espaciales 2D. El parámetro que suele ocuparse es el tamaño del sector en que se va a aplicar la operación de agrupamiento, este se llama *pool_size*.

La tercera capa distinta es la capa de agrupación *GlobalAveragePooling2D* y simplemente realiza la operación de agrupación promedio global para datos espaciales 2D. El parámetro a utilizar es el mismo que el de *MaxPooling2D*.

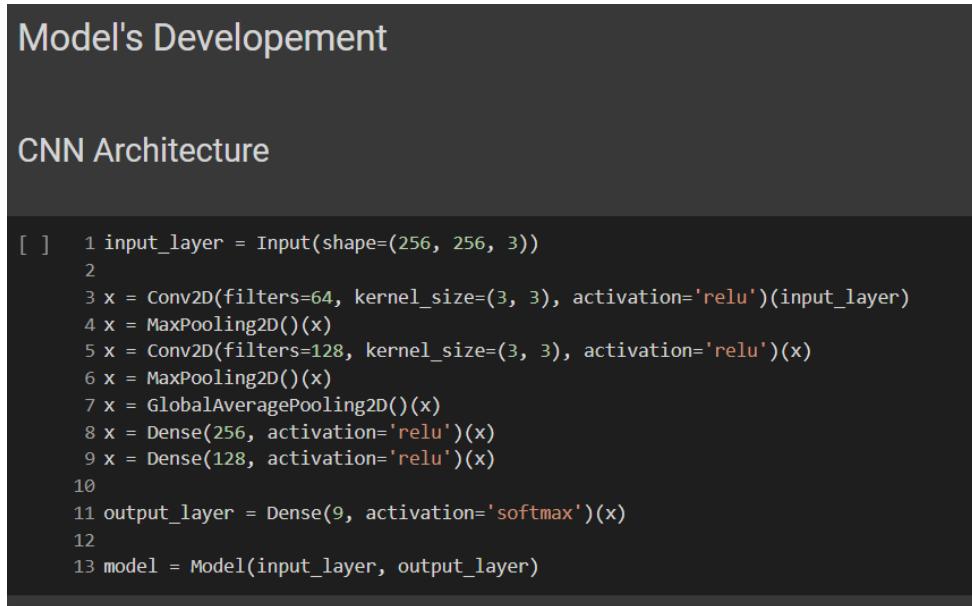
Por último, la cuarta capa diferente es simplemente una capa de red neuronal densamente condensada y se llama *Dense*. El primer parámetro de estas capas es la cantidad de neuronas de la capa neuronal y el segundo es la función de activación.

Cabe mencionar que las capas convolucionales *Conv2D* y las capas de redes neuronales *Dense* usan capas de activación; estas capas aplican la función de activación *RELU*, la función lineal rectificada. En la función RELU, si el valor es < 0 , el resultado siempre será 0; si el valor es > 0 , el valor se mantiene y el resultado tiende al ∞ .

Para la última capa, la capa de salida, la función de activación a utilizar es la función Softmax. La función Softmax da la probabilidad de cada una de las posibles salidas, es por eso que se utiliza para conocer la probabilidad de salida en las clasificaciones multiclas.

Finalmente, se llega a la clase *Model*. Esta clase se encarga de agrupar las capas en un objeto con funciones de entrenamiento e inferencia, en otras palabras, esta clase crea un modelo solo especificando las entradas y las salidas.

La arquitectura del modelo de clasificación se muestra en la Imagen 4.80.



```
[ ]    1 input_layer = Input(shape=(256, 256, 3))
      2
      3 x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(input_layer)
      4 x = MaxPooling2D()(x)
      5 x = Conv2D(filters=128, kernel_size=(3, 3), activation='relu')(x)
      6 x = MaxPooling2D()(x)
      7 x = GlobalAveragePooling2D()(x)
      8 x = Dense(256, activation='relu')(x)
      9 x = Dense(128, activation='relu')(x)
     10
     11 output_layer = Dense(9, activation='softmax')(x)
     12
     13 model = Model(input_layer, output_layer)
```

Imagen 4.80 Arquitectura de la CNN para la clasificación de mariscos

4.5.5.2 Resumen del Modelo

Después de haber diseñado el modelo de clasificación, es momento de analizar la cantidad de parámetros totales en la red neuronal convolucional.

```
[ ] 1 model.summary()

Model: "model"
=====
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 256, 256, 3]	0
conv2d (Conv2D)	(None, 254, 254, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 127, 127, 64)	0
conv2d_1 (Conv2D)	(None, 125, 125, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 256)	33024
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 9)	1161

```
=====
Total params: 142,729
Trainable params: 142,729
Non-trainable params: 0
```

Imagen 4.81 Resumen de la CNN para clasificación de mariscos

Lo primero a tomar en consideración en el análisis de la arquitectura, es conocer el tamaño de entrada y este es:

$$[(None, 256, 256, 3)]$$

En el tamaño de entrada hay un *None* y Keras lo usa para representar el hecho de que es posible pasar cualquier cantidad de imágenes a través de la red de forma simultánea. Como el modelo solo utiliza álgebra de tensores, no es necesario pasar imágenes a través del modelo de forma individual, sino en conjunto, como un lote.

Después, se le aplica la primera capa convolucional con un tamaño de filtro (3, 3, 3), donde los primeros dos treses representan la altura y el ancho del filtro, mientras que el último tres es la cantidad de canales de la capa anterior, puesto que son los canales de entrada de las imágenes (rojo, verde y azul).

Entonces, el número de parámetros, o pesos, es:

$$(3^3 + 1) \times 64 = 1792$$

Donde el $+1$ se debe a la inclusión de un término de sesgo adjunto a cada uno de los filtros. También hay que considerar que la profundidad de los filtros en una capa es siempre la misma que el número de canales en la capa anterior.

Para determinar el tamaño de salida de una capa convolucional con un tamaño de filtro (3,3) es:

$$(None, altura - 2, ancho - 2, filtros)$$

Por lo que la salida de la primera capa convolucional es de tamaño:

$$(None, 254, 254, 64)$$

La siguiente capa, es una capa de agrupamiento, por lo que no da parámetros, sin embargo sí reduce el tamaño de salida de la capa. En este caso, la altura y el ancho se dividen por la mitad, por lo que la salida de la primera capa de agrupamiento *MaxPooling2D* es:

$$(None, 127, 127, 64)$$

En la segunda capa convolucional se tiene un filtro (3,3,64), entonces, el número de parámetros es:

$$(3^2 \times 64 + 1) \times 128 = 73,856$$

La salida de la segunda capa convolucional es:

$$(None, 125, 125, 128)$$

Después, viene una segunda capa de agrupamiento y en este caso, la altura y el ancho tienen como valor un número impar; por lo que el resultado de salida se le deberá aplicar la función piso, es decir, al tener 125 como valor, este se divide entre 2 y se obtiene 62.5, pero al aplicar la función piso, se obtiene solamente el 62. En resumen, la salida de la segunda capa de agrupamiento *MaxPooling2D* es:

$(None, 62, 62, 128)$

La siguiente capa es una capa de agrupamiento global llamada *GlobalAveragePooling2D*; esta capa al ser de agrupamiento, tampoco deja parámetro alguno, sin embargo el tamaño de salida también cambia y solo deja como valor el número total de filtros, por lo que la salida de esta capa de agrupamiento es:

$(None, 128)$

Posteriormente, la capa siguiente es una capa *Dense* y para calcular sus parámetros solamente se multiplica:

$$\text{valor de neuronas} \times (\text{valor antecesor} + 1)$$

Entonces, el número de parámetros de la primera capa *Dense* es:

$$256 \times 129 = 33,024$$

Y la salida de las capas *Dense* es el número de neuronas que posee esa misma capa, por lo que la salida de la primera capa *Dense* es:

$(None, 256)$

El número de parámetros de la segunda capa *Dense* es:

$$128 \times 257 = 32,896$$

La salida de la segunda capa *Dense* es:

$(None, 128)$

Finalmente, la última capa *Dense* que además, es la salida del modelo; tiene por número de parámetros el:

$$9 \times 129 = 1,161$$

La salida final del modelo queda de la siguiente forma: $(None, 9)$.

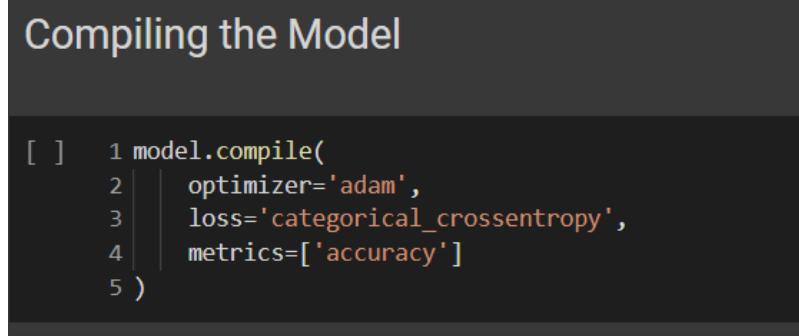
Y el número total de parámetros es: 142,729.

4.5.5.3 Compilado del Modelo

El modelo de la CNN se compila con el optimizador Adam, es decir, que el optimizador que implementa el algoritmo Adam. Esta optimización es un método de descenso del gradiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden.

El parámetro *loss* es la función de pérdida. Para el modelo se usa la pérdida *categorical_crossentropy* y esta lo calcula entre las etiquetas y las predicciones. Esta función se usa cuando hay dos o más clases de etiquetas. Las etiquetas se proporcionan en la representación *one_hot*.

El último parámetro es una lista de métricas que el modelo evaluará durante el entrenamiento y la comprobación.



```
[ ] 1 model.compile(
2     optimizer='adam',
3     loss='categorical_crossentropy',
4     metrics=['accuracy']
5 )
```

Imagen 4.82 Compilado del Modelo con optimizador Adam

4.5.5.4 Entrenamiento del Modelo

Para entrenar el modelo se usa el método *fit()*. Este método lo entrena para un número fijo de épocas o iteraciones en el dataset.

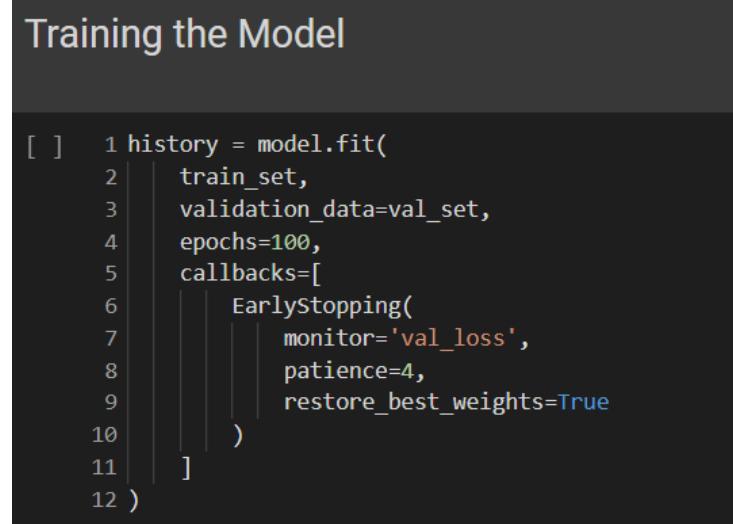
El primer argumento hace referencia a los datos de entrada, en este caso el *data_train*.

El argumento *validation_data* es sobre aquellos datos con que se van a evaluar la pérdida y cualquier otra métrica al final de cada época.

El argumento de *epochs* tiene que ser un número entero.

El último argumento es *callbacks* y se emplea el método *EarlyStopping*; este módulo detiene el entrenamiento cuando una métrica monitoreada haya dejado de mejorar.

El argumento *monitor* es la métrica a monitorear. El argumento *patience* es el número de épocas en las que se detendrá el entrenamiento si la métrica no mejora. El último parámetro *restore_best_weights*, restaura los pesos del modelo de la época con el mejor valor de la métrica monitoreada.



```
[ ] 1 history = model.fit(  
2     | train_set,  
3     | validation_data=val_set,  
4     | epochs=100,  
5     | callbacks=[  
6         |     EarlyStopping(  
7             |         | monitor='val_loss',  
8             |         | patience=4,  
9             |         | restore_best_weights=True  
10            |     )  
11        ]  
12 )
```

Imagen 4.83 Entrenamiento del Modelo

4.5.6 Código en Kaggle

El código elaborado en la plataforma de Kaggle es prácticamente el mismo que en Colab, sin embargo hay pequeñas diferencias.

La primera diferencia es que no se debe clonar un repositorio o crear un dataset para realizar el modelo, porque todos los datasets ya se encuentran en la misma plataforma.

Para agregar el dataset al Notebook, se da clic en el botón *Add data* del menú ubicado en la parte derecha.

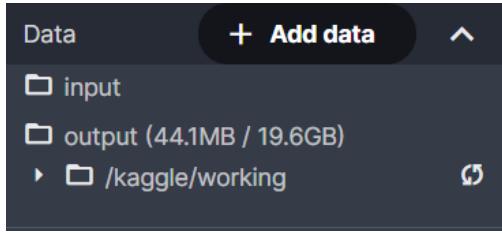


Imagen 4.84 Botón *Add data* del menú

Después en el cuadro de diálogo que aparece, se busca en la barra de búsqueda el dataset llamado: *A Large Scale Fish Dataset*. En este caso, aparece en la segunda opción de datasets, por lo que simplemente con seleccionar *Add*, se agrega el dataset al Notebook.

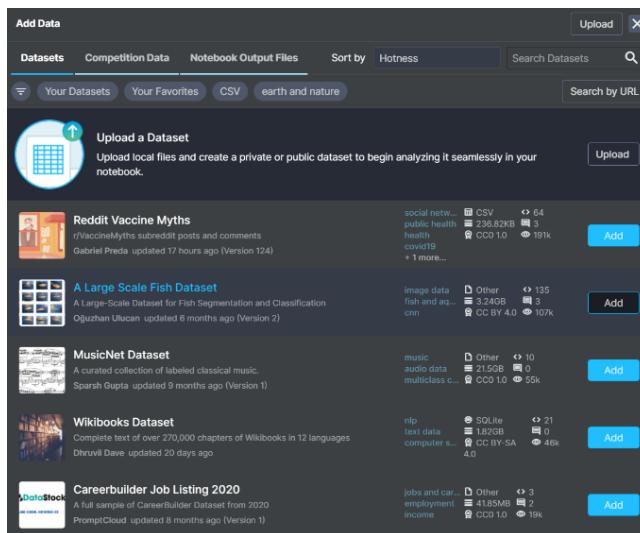


Imagen 4.85 Cuadro de diálogo para agregar un dataset al Notebook

La segunda diferencia es la forma de importar librerías en Kaggle, sino marca un error.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Dense, Conv2D, GlobalAveragePooling2D, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping
```

Imagen 4.86 Importación de Librerías en Kaggle

La tercera diferencia es el uso del módulo Path de la librería pathlib para ubicar la ruta del dataset, así como la forma de obtener las rutas de los archivos, pues estos se obtienen usando la librería glob y con un código que permite encontrar todos los archivos que tengas extensión .png dentro de las subrutas o subcarpetas.

```
# Dataset Path
dataset_path = Path('../input/a-large-scale-fish-dataset/Fish_Dataset/Fish_Dataset')

# Getting Filepaths
filepaths = list(dataset_path.glob(r'**/*.png'))
```

Imagen 4.87 Obtención de las rutas de archivos con Kaggle

Cabe mencionar que las rutas de los archivos son objetos PosixPath y no strings como en Colab.

```
filepaths
```

```
[PosixPath('../input/a-large-scale-fish-dataset/Fish_Dataset/Hourse Mackerel/Hourse Mackerel/00929.png'),
 PosixPath('../input/a-large-scale-fish-dataset/Fish_Dataset/Hourse Mackerel/Hourse Mackerel/00704.png'),
 PosixPath('../input/a-large-scale-fish-dataset/Fish_Dataset/Hourse Mackerel/Hourse Mackerel/00562.png'),
 PosixPath('../input/a-large-scale-fish-dataset/Fish_Dataset/Hourse Mackerel/Hourse Mackerel/00237.png'),
 PosixPath('../input/a-large-scale-fish-dataset/Fish_Dataset/Hourse Mackerel/Hourse Mackerel/00406.png'),
```

Imagen 4.88 Primeros cinco ejemplos de la obtención de rutas de archivos en Kaggle

La cuarta y última diferencia tiene que ver con las rutas de los archivos. Al ser estos no un string, no es posible generar una *Series* de Pandas con ellos; por lo que al final de estar generando la *Serie*, se agrega el método *astype(str)* para que se conviertan a strings los datos.

```
# Filepaths/Labels Series generation
filepaths = pd.Series(filepaths, name='Filepath').astype(str)
labels = pd.Series(labels, name='Label')
```

Imagen 4.89 Generación de Series con Pandas con Kaggle

4.6 Principios de Azure y Custom Vision

Para crear, entrenar y probar un modelo de clasificación, sin código alguno, mediante la API de Custom Vision, tener una suscripción a Azure es imprescindible.

4.6.1 Creación del Proyecto en Custom Vision Service

En primer lugar se debe iniciar sesión en el portal de Custom Vision Service. Se puede acceder desde un navegador o con el siguiente enlace:

<https://www.customvision.ai/>

Después de haber accedido, se selecciona *New Project* para crear un proyecto.

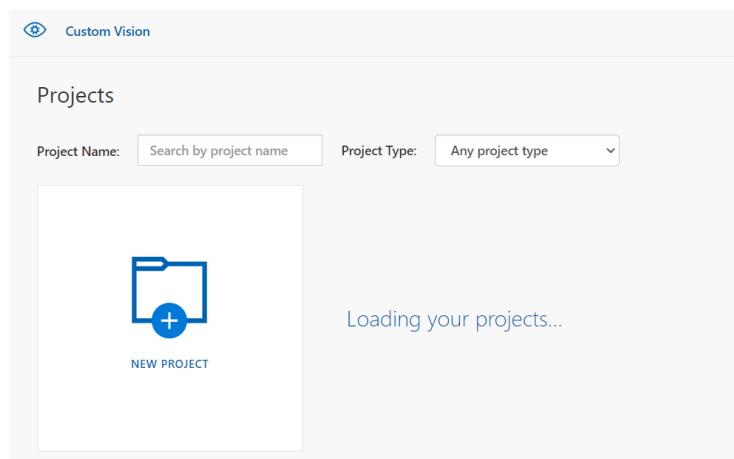


Imagen 4.90 Sección *Proyectos* de Custom Vision Service

En el cuadro de diálogo *Create new project*, se escribe el nombre del proyecto y una breve descripción.

Create new project		X
Name* <input type="text" value="Enter project name"/>		
Description <input type="text" value="Enter project description"/>		
Resource <input type="text" value="- Please Choose -"/>		create new
<input type="button" value="Cancel"/>		<input type="button" value="Create project"/>

Imagen 4.91 Cuadro de diálogo *Create new project*

El nombre del proyecto es *Seafood Classifier* y la descripción es: “Predict 9 types of turkish seafood with Microsoft Custom Vision Service”.

Después se selecciona un *Recurso* para el proyecto, este es una *Cuenta de varios servicios de Cognitive Services*. Sino se cuenta con uno, se selecciona *create new* para crear uno nuevo.

Create New Resource

Name*
Enter resource name

Subscription*
Azure para estudiantes

Resource Group*
Choose a resource group [create new](#)

Kind
CognitiveServices

Location
South Central US

Pricing Tier
S0

[Create resource](#)

Imagen 4.92 Cuadro de diálogo *Create New Resource*

Después se selecciona un *Grupo de Recursos* para el *Recurso*. Sino se cuenta con uno se selecciona *create new* para crear uno nuevo.

Create New Resource Group

Name*
tsu_project

Location
South Central US

[Create resource group](#)

Imagen 4.93 Cuadro de diálogo *Create New Resource Group*

El nombre del *Grupo de Recursos* es *tsu_project* y la *Locación* es *South Central US*, ya que es la región de Azure más cercana a México. Sin embargo, en los próximos meses puede que ya se pueda seleccionar una región de Azure aquí en México, para tomarlo en consideración.

Después de tener ya un *Grupo de Recursos* disponible, se completan los datos. El nombre del *Recurso* es *seafood_segmenter_classifier*, el *Tipo* de recurso es un *AllCognitiveServices* y la *Locación* de la región de Azure tiene que ser la misma que la que se usó en el *Grupo de Recursos*. Por último se selecciona el *Nivel de Precios* que es estándar y gratuito (*S0*).

Create New Resource

Name*
seafood_segmenter_classifier

Subscription*
Azure para estudiantes

Resource Group* [create new](#)
tsu_project

Kind
CognitiveServices

Location
South Central US

Pricing Tier
S0

[Create resource](#)

Imagen 4.94 Cuadro de diálogo *Create New Resource*
con los datos completos

Tras seleccionar el *Grupo de Recursos*, se seleccionan:

- *Tipo de Proyecto*: Clasificación
- *Tipo de Clasificación*: Multiclasificación (Etiqueta única por imagen)
- *Dominios*: General [A1]

Se escoge el Dominio General [A1] porque está optimizado para una mayor precisión, además que es recomendado para datasets muy grandes o escenarios de usuario más difíciles, puesto que requieren más tiempo de entrenamiento.

Create new project X

Name*

Description

Resource* create new

[Manage Resource Permissions](#)

Project Types (1)

Classification
 Object Detection

Classification Types (1)

Multilabel (Multiple tags per image)
 Multiclass (Single tag per image)

Domains:

General [A2]
 General [A1]
 General
 Food
 Landmarks
 Retail
 General (compact) [S1]
 General (compact)
 Food (compact)
 Landmarks (compact)
 Retail (compact)

Pick the domain closest to your scenario. Compact domains are lightweight models that can be exported to iOS/Android and other platforms. [Learn More](#)

Cancel Create project

Imagen 4.95 Cuadro de diálogo *Create new project* con los datos completos

Para crear el proyecto se selecciona *Create project*.

4.6.2 Carga de Imágenes Etiquetadas

En el proyecto *Seafood Classifier*, al seleccionar el signo más en el menú de la izquierda y a la derecha de *Etiquetas*, se pueden crear etiquetas nuevas.

El cuadro de diálogo que se muestra es *Create a new tag* y en él, se pueden añadir todas las etiquetas para organizar y buscar las imágenes. Para añadir cada etiqueta, solo se escribe la etiqueta a añadir y se selecciona en *Guardar*.

Create a new tag X

You will use tags to organize and search your images.

Tag Name

Is Negative?

Save

Imagen 4.96 Cuadro de diálogo para añadir etiquetas

En la Imagen 4.97 se ve una lista de las 9 etiquetas que se necesitan para el proyecto.

Tags		+
Tagged Untagged		
Showing: all tagged images		
Search For Tags:		
<input type="text"/> <input type="checkbox"/> Black Sea Sprat 0 ... <input type="checkbox"/> Gilt-Head Bream 0 ... <input type="checkbox"/> Horse Mackerel 0 ... <input type="checkbox"/> Red Mullet 0 ... <input type="checkbox"/> Red Sea Bream 0 ... <input type="checkbox"/> Sea Bass 0 ... <input type="checkbox"/> Shrimp 0 ... <input type="checkbox"/> Striped Red Mullet 0 ... <input type="checkbox"/> Trout 0 ... 		

Imagen 4.97 Lista de etiquetas, sin imagen alguna

Para agregar las imágenes a su respectiva etiqueta, se selecciona *Add images*. Esta opción es la primera de las que se encuentran en el medio.

Como primer ejemplo, se busca la carpeta *Black Sea Sprat* dentro de la computadora local, para agregar las 1,000 imágenes del dataset. Después de que las imágenes sean reconocidas para subirse a la plataforma, se selecciona su respectiva etiqueta en la sección *My Tags*. Luego, ya es posible subir las imágenes, por lo que se selecciona el botón *Upload 1000 files*.

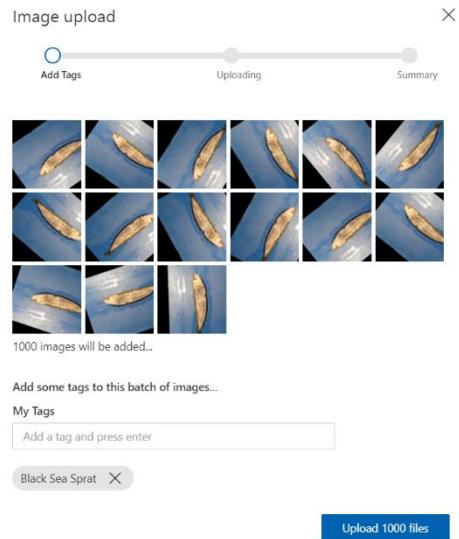


Imagen 4.98 Cuadro de diálogo *Image upload*

Finalmente, las 1000 imágenes fueron cargadas exitosamente.

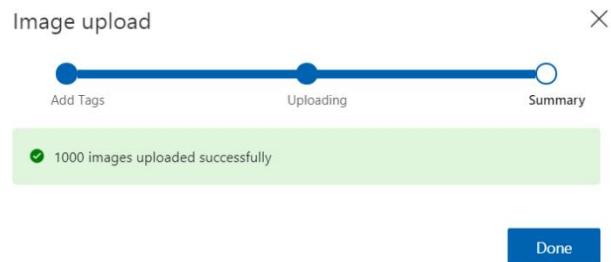


Imagen 4.99 Cuadro de diálogo de carga exitosa de las imágenes

Se repite este proceso anterior para cada una de las etiquetas restantes y el resultado es el siguiente:

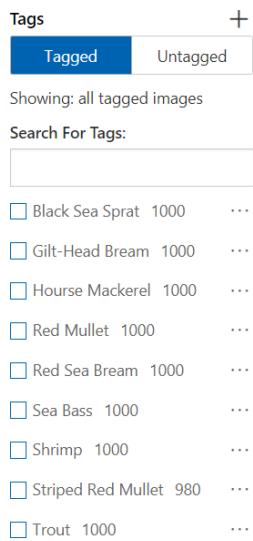


Imagen 4.100 Lista de etiquetas con datos

4.6.3 Entrenamiento del Modelo

En esta sección se entrena el modelo con las imágenes cargadas y etiquetadas en la sección anterior, la Sección 4.6.2.

Lo primero que se hace es seleccionar el botón verde *Train* situado en la parte superior de la página para entrenar el modelo. Cada vez que se entrena el modelo, se crea una nueva iteración. Custom Vision Service mantiene las iteraciones, lo que permite comparar a través del tiempo su progreso.

Después, sale un cuadro de diálogo en el que se dan dos opciones a escoger para el tipo de entrenamiento. Los entrenamientos avanzados cuestan con respecto al tiempo que se tiene entrenando al modelo, sin embargo, al ser este un modelo sencillo, la primera opción es más que suficiente. Esta primera opción es un entrenamiento rápido, es gratuito y sirve adecuadamente.

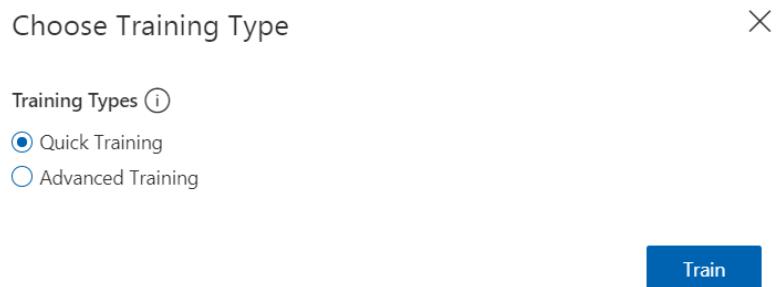


Imagen 4.101 Cuadro de diálogo de opciones de entrenamiento

Finalmente, solo queda esperar a que se complete el proceso de entrenamiento. Al ser un modelo de entrenamiento con una considerable cantidad de datos, se tiene que esperar un poco tiempo para que finalice este entrenamiento rápido del modelo. El tiempo estimado es de 40 minutos.

CAPÍTULO 5

PRUEBAS Y RESULTADOS

5.1 Resultados del Modelo CNN

Tras haber entrenado una CNN con una estructura basada en la API funcional de Keras, API cuyo backend es TensorFlow, los resultados son:

5.1.1 Resultados del Entrenamiento

Después de haber mandado a entrenar el modelo CNN para la clasificación de mariscos se observa que las primeras 5 épocas del modelo incrementan rápido su precisión y con valores similares, tanto con el dataset de entrenamiento y el de validación. Igualmente la pérdida desciende en ambos casos de manera progresiva.

```

Epoch 1/100
210/210 [=====] - 137s 500ms/step - loss: 1.9523 - accuracy: 0.2456 - val_loss: 1.6440 - val_accuracy: 0.3667
Epoch 2/100
210/210 [=====] - 91s 433ms/step - loss: 1.5210 - accuracy: 0.4310 - val_loss: 1.3472 - val_accuracy: 0.4933
Epoch 3/100
210/210 [=====] - 89s 425ms/step - loss: 1.2419 - accuracy: 0.5337 - val_loss: 1.1988 - val_accuracy: 0.5711
Epoch 4/100
210/210 [=====] - 91s 432ms/step - loss: 0.9794 - accuracy: 0.6410 - val_loss: 0.8452 - val_accuracy: 0.6830
Epoch 5/100
210/210 [=====] - 90s 428ms/step - loss: 0.7559 - accuracy: 0.7165 - val_loss: 0.6535 - val_accuracy: 0.7637

```

Imagen 5.1 Resultado de las primeras 5 épocas de entrenamiento del Modelo CNN

El modelo entrenó un total 29 épocas, después que no hubiera una mejora en los últimos cuatro valores de la pérdida con los datos de validación (*val_loss*), logrando así evitar que el modelo genere overfitting. El overfitting es cuando el modelo encaja perfectamente con los datos de entrenamiento. El underfitting es cuando el modelo no es capaz de lograr una relación entre los datos de entrada y los de salida de una manera precisa. Un buen modelo no debe tener estos dos conceptos antes mencionados.

```

Epoch 24/100
210/210 [=====] - 91s 432ms/step - loss: 0.1358 - accuracy: 0.9508 - val_loss: 0.1257 - val_accuracy: 0.9526
Epoch 25/100
210/210 [=====] - 90s 428ms/step - loss: 0.1467 - accuracy: 0.9519 - val_loss: 0.1111 - val_accuracy: 0.9659
Epoch 26/100
210/210 [=====] - 89s 422ms/step - loss: 0.0942 - accuracy: 0.9705 - val_loss: 0.1205 - val_accuracy: 0.9570
Epoch 27/100
210/210 [=====] - 89s 422ms/step - loss: 0.1072 - accuracy: 0.9629 - val_loss: 0.1219 - val_accuracy: 0.9615
Epoch 28/100
210/210 [=====] - 89s 422ms/step - loss: 0.0948 - accuracy: 0.9683 - val_loss: 0.1123 - val_accuracy: 0.9578
Epoch 29/100
210/210 [=====] - 89s 423ms/step - loss: 0.1254 - accuracy: 0.9541 - val_loss: 0.1813 - val_accuracy: 0.9296

```

Imagen 5.2 Resultado de las últimas 5 épocas de entrenamiento del Modelo CNN

5.1.2 Evaluación del Modelo

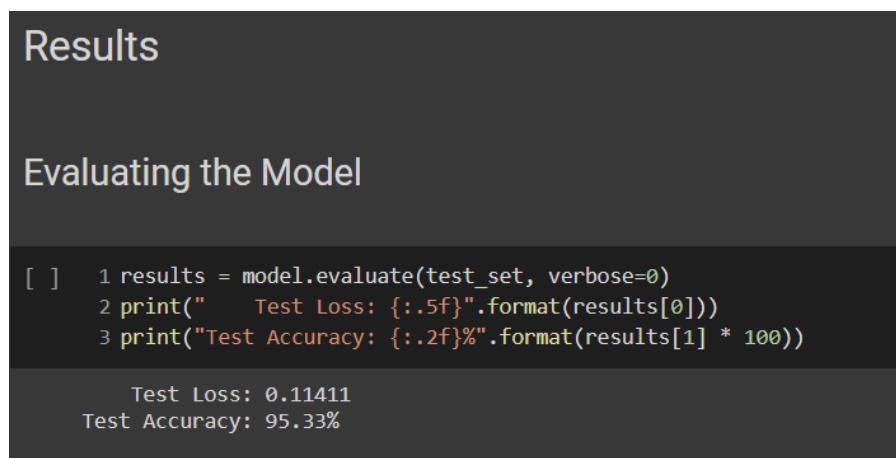
Para evaluar el modelo, se usa el método *evaluate* del módulo *Model*, módulo que se ha utilizado desde que se desarrolló la arquitectura de la CNN.

Este método retorna los valores de pérdida y los valores de métricas del modelo para el modo de comprobación.

El primer argumento hace referencia a los datos de entrada; como se necesitan datos para comprobación, el *test_set* es el set adecuado y hecho para este propósito.

El segundo argumento es *verbose=0*, este argumento solo tiene a 0 o 1 como valores disponibles. Este brinda la opción de mostrar una barra de progreso durante el proceso de evaluación, sin embargo como no la quiero mostrar, por eso es 0, para que esté en un modo silencioso.

La primera línea de impresión es para obtener la pérdida durante la comprobación; y la segunda línea es para obtener el porcentaje de precisión que tuvo durante este proceso.



```

Results

Evaluating the Model

[ ] 1 results = model.evaluate(test_set, verbose=0)
2 print("    Test Loss: {:.5f}".format(results[0]))
3 print("Test Accuracy: {:.2f}%".format(results[1] * 100))

    Test Loss: 0.11411
Test Accuracy: 95.33%

```

Imagen 5.3 Evaluación del Modelo

Como se observa en las impresiones, el modelo obtuvo en las comprobaciones, una pérdida de 0.114; y una precisión del 95.33%.

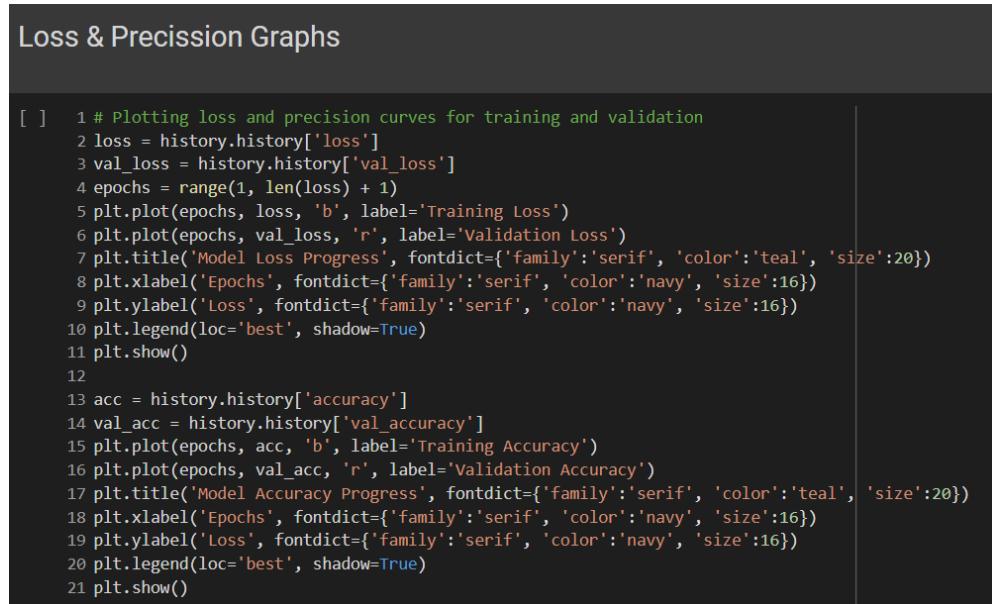
5.1.3 Gráficas de Pérdida y Precisión

Estás gráficas se elaboraron de la manera tradicional de llamar a una gráfica con *pyplot*, más que nada para que visualmente sean llamativas y claras. También se usaron muchos tipos de parámetros en los distintos métodos, para que esto sirviera de práctica en la visualización de datos con esta librería.

Este código elaboró dos gráficas, una para las curvas de pérdida y otra para las curvas de precisión.

Las gráficas tienen en el eje *x* a las épocas como escala. La primera gráfica tiene al eje *y* como la función de pérdida, mientras que la segunda gráfica tiene al eje *y* como la precisión adquirida.

En ambas gráficas se usaron funciones para mostrar las etiquetas de ambos ejes y parámetros para la personalización de las tipografías, como el tipo de fuente, el tamaño y el color.



```
[ ] 1 # Plotting loss and precision curves for training and validation
2 loss = history.history['loss']
3 val_loss = history.history['val_loss']
4 epochs = range(1, len(loss) + 1)
5 plt.plot(epochs, loss, 'b', label='Training Loss')
6 plt.plot(epochs, val_loss, 'r', label='Validation Loss')
7 plt.title('Model Loss Progress', fontdict={'family':'serif', 'color':'teal', 'size':20})
8 plt.xlabel('Epochs', fontdict={'family':'serif', 'color':'navy', 'size':16})
9 plt.ylabel('Loss', fontdict={'family':'serif', 'color':'navy', 'size':16})
10 plt.legend(loc='best', shadow=True)
11 plt.show()
12
13 acc = history.history['accuracy']
14 val_acc = history.history['val_accuracy']
15 plt.plot(epochs, acc, 'b', label='Training Accuracy')
16 plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
17 plt.title('Model Accuracy Progress', fontdict={'family':'serif', 'color':'teal', 'size':20})
18 plt.xlabel('Epochs', fontdict={'family':'serif', 'color':'navy', 'size':16})
19 plt.ylabel('Loss', fontdict={'family':'serif', 'color':'navy', 'size':16})
20 plt.legend(loc='best', shadow=True)
21 plt.show()
```

Imagen 5.4 Código para trazar Gráficos de Pérdida y Precisión

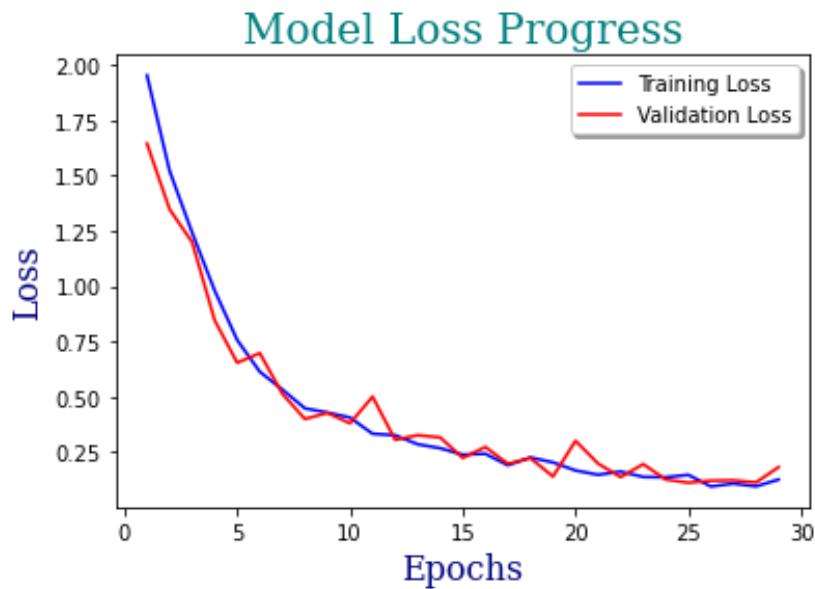


Imagen 5.5 Gráfica de Pérdida del Modelo CNN

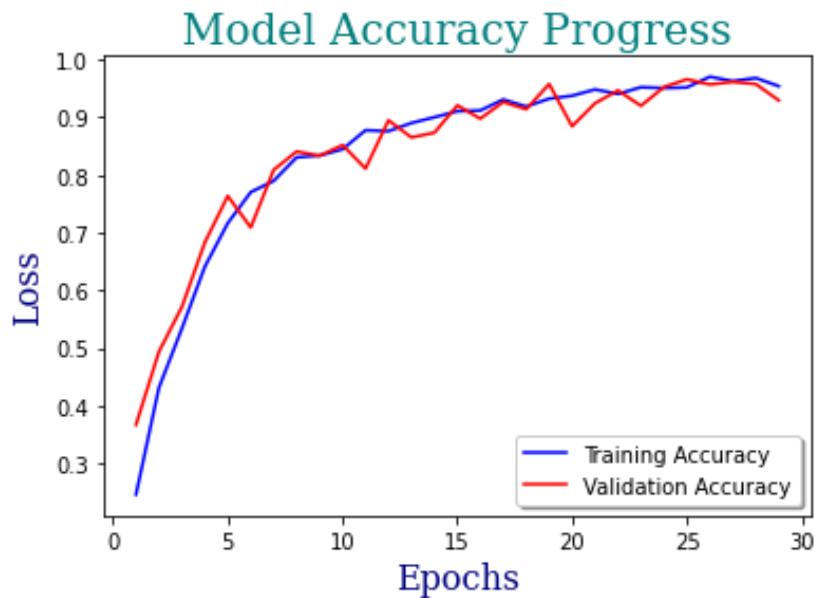


Imagen 5.6 Gráfica de Precisión del Modelo CNN

Tras ver las gráficas, se nota que las curvas de pérdida y las curvas de precisión son decentes ya que mantienen un decrecimiento y un crecimiento progresivo, pero lo que más se logra apreciar es que van de la mano las curvas porque no hay tanta diferencia en los valores de entrenamiento y validación en cada época.

5.1.4 Reporte de Clasificación y Matriz de Confusión

En esta sección se visualizan el reporte de clasificación y la matriz de confusión del entrenamiento del modelo. Para poder visualizarlos se hace uso de la librería *sklearn* y del módulo *metrics*. Este módulo incluye métricas relacionadas con la estadística.

La primera métrica a utilizar es un reporte de clasificación, cuya función es crear un informe de texto que muestre las principales métricas de clasificación. El primer parámetro es un array en el que están los valores correctos o valores verdaderos; el segundo parámetro son las predicciones y estos son los objetivos estimados en el clasificador; por último, el tercer argumento muestra los nombres que coinciden con las etiquetas.

El segundo tipo de métrica es una matriz de confusión el cual evalúa la precisión de un clasificador. El primer argumento es para los valores verdaderos del set; mientras que el segundo argumento son las predicciones de dichos datos. Para la representación de esta matriz se usó un módulo de seaborn llamado *heatmap*, que prácticamente es una matriz de confusión, solo que con otro nombre.

```
Classification Report and Confusion Matrix

[ ] 1 predictions = np.argmax(model.predict(test_set), axis=1)
2
3 cm = confusion_matrix(test_set.labels, predictions)
4 clr = classification_report(test_set.labels, predictions, target_names=list(train_set.class_indices.keys()))
5
6 plt.figure(figsize=(18, 18))
7 sns.heatmap(cm, annot=True, annot_kws={"size": 12}, fmt='g', vmin=0, cmap='GnBu', cbar=False, linewidths=0.1, linecolor='teal')
8
9 plt.title("Confusion Matrix", fontdict={'family':'serif', 'color':'teal', 'size':20})
10 plt.xticks(ticks=np.arange(9) + 0.5, labels=list(train_set.class_indices.keys()), family='serif', size=10)
11 plt.yticks(ticks=np.arange(9) + 0.5, labels=list(train_set.class_indices.keys()), family='serif', size=10, verticalalignment="center")
12 plt.xlabel("Predicted", fontdict={'family':'serif', 'color':'navy', 'size':16})
13 plt.ylabel("Actual", fontdict={'family':'serif', 'color':'navy', 'size':16})
14 plt.show()
15
16 print("Classification Report:\n-----\n", clr)
```

Imagen 5.7 Código del Reporte de Clasificación y la Matriz de Confusión

La precisión, o *precision*, es el porcentaje de predicciones correctamente clasificadas.

La recuperación, o *recall*, es el porcentaje de valores reales correctamente clasificados.

El F1-score es la media armónica entre la precisión y la recuperación.

El soporte, o *support*, es la cantidad de ocurrencias de cada clase en el conjunto de valores deseados.

Accuracy, o precisión, es la precisión que tiene el clasificador en general.

Macro average, o promedio general, es el promedio de las puntuaciones de todas las clases.

Weighted average, o media ponderada, es la suma de todas las clases después de multiplicar sus respectivas proporciones de clases.

Classification Report:				
	precision	recall	f1-score	support
Black Sea Sprat	0.99	1.00	1.00	154
Gilt-Head Bream	0.87	0.89	0.88	148
Hourse Mackerel	0.99	0.97	0.98	156
Red Mullet	1.00	0.99	1.00	162
Red Sea Bream	0.88	0.96	0.92	139
Sea Bass	0.90	0.94	0.91	155
Shrimp	1.00	1.00	1.00	147
Striped Red Mullet	0.99	0.99	0.99	156
Trout	0.96	0.81	0.88	133
accuracy			0.95	1350
macro avg	0.95	0.95	0.95	1350
weighted avg	0.95	0.95	0.95	1350

Imagen 5.8 Reporte de Clasificación del Modelo CNN

		Confusion Matrix								
		Black Sea Sprat	Gilt-Head Bream	Horse Mackerel	Red Mullet	Red Sea Bream	Sea Bass	Shrimp	Striped Red Mullet	Trout
Actual		154	0	0	0	0	0	0	0	0
Black Sea Sprat		0	131	0	0	3	9	0	0	5
Gilt-Head Bream		0	152	0	0	0	4	0	0	0
Horse Mackerel		0	0	161	0	0	0	0	1	0
Red Mullet		0	0	0	134	0	0	0	0	0
Red Sea Bream		0	5	0	0	145	0	0	0	0
Sea Bass		0	6	0	0	4	147	0	0	0
Shrimp		0	0	0	0	0	0	155	0	0
Striped Red Mullet		1	0	0	0	0	0	0	108	0
Trout		0	8	2	0	11	4	0	0	108
		Black Sea Sprat	Gilt-Head Bream	Horse Mackerel	Red Mullet	Red Sea Bream	Sea Bass	Shrimp	Striped Red Mullet	Trout
Predicted										

Imagen 5.9 Matriz de Confusión del Modelo CNN

Para finalizar con esta sección, la línea de código para guardar el modelo es:

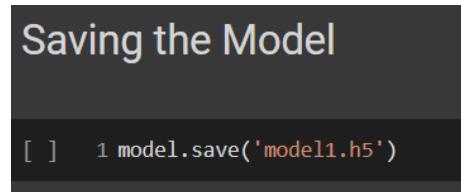


Imagen 5.10 Línea de Código para
guardar el Modelo CNN

5.2 Resultados del Modelo con Custom Vision Service

Tras haber entrenado un clasificador multiclase de mariscos en la plataforma de Custom Vision Services, usando a Custom Vision como la API, los resultados son:

5.2.1 Resultados del Entrenamiento

Luego de esperar a que se completara el proceso de entrenamiento del clasificador, en la pestaña de *Rendimiento* se encuentran las estadísticas de la primera iteración.

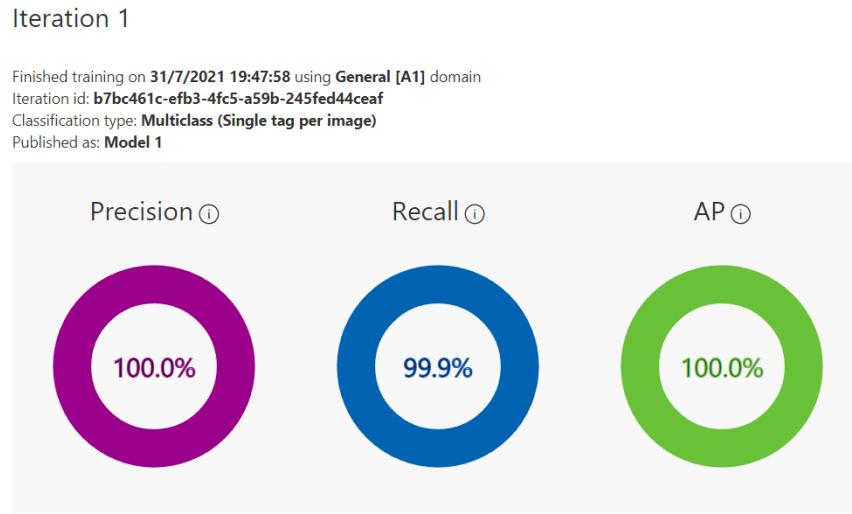


Imagen 5.11 Estadísticas de entrenamiento de la Iteración 1

Los resultados muestran dos medidas de precisión del Modelo:

- *Precision*; este número dice: si el modelo predice una etiqueta, ¿qué probabilidades hay de que sea correcto.
- *Recall*; este número dice: de las etiquetas que debería predecirse, ¿qué porcentaje se encontraron correctamente en el modelo?
- *AP*; es una medida del rendimiento del modelo, pues resume la *Precision* y el *Recall* en diferentes umbrales.

Además, se muestra una tabla con el rendimiento de cada categoría o de cada etiqueta, por separado:

Performance Per Tag

Tag	Precision	\wedge	Recall	A.P.	Image count
Trout	100.0%		100.0%	100.0%	1000 
Striped Red Mullet	100.0%		100.0%	100.0%	980 
Shrimp	100.0%		100.0%	100.0%	1000 
Sea Bass	100.0%		99.5%	100.0%	1000 
Red Sea Bream	100.0%		100.0%	100.0%	1000 
Red Mullet	100.0%		100.0%	100.0%	1000 
Hourse Mackerel	100.0%		100.0%	100.0%	1000 
Gilt-Head Bream	100.0%		100.0%	100.0%	1000 
Black Sea Sprat	100.0%		100.0%	100.0%	1000 

Imagen 5.12 Rendimiento de cada Etiqueta

5.2.2 Prueba del Modelo

Ahora que ya se tiene entrenado el modelo, se puede probar el clasificador de una manera muy sencilla. Para ello, en la parte superior de la página, se selecciona *Quick Test*.



Imagen 5.13 Parte superior de la página, incluido *Quick Test*

5.2.2.1 Prueba Rápida (*Quick Test*)

Después, se abre un cuadro de diálogo en donde se pide ingresar la URL de una imagen o solamente subir una imagen desde el dispositivo local.

En este primer caso, es un caso atípico debido a que el pez analizado es un Betta Cola de Corona, una especie totalmente distinta a las categorías previamente analizadas; por lo que

el clasificador intentará predecir el resultado con aquellas categorías que se tienen. Según el clasificador, el pez con quien tiene más similitudes es con el mullet rojo (*Red Mullet*).

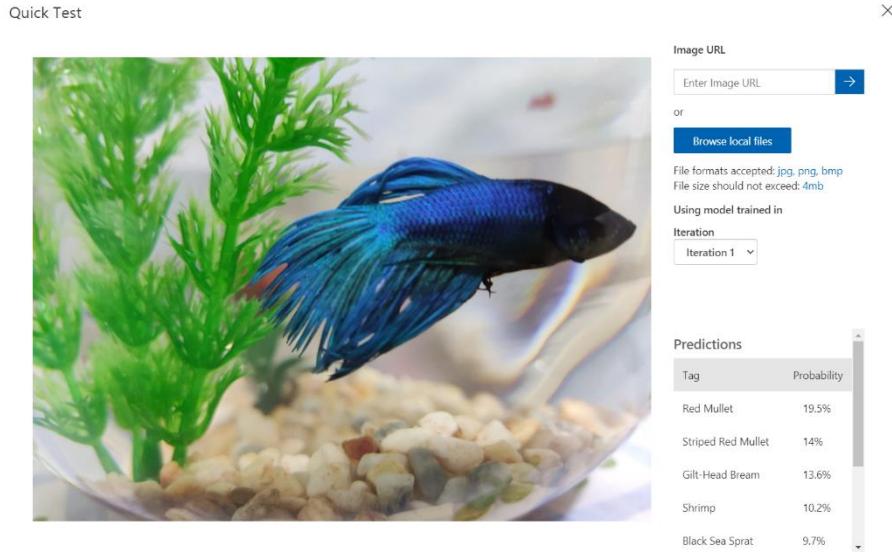


Imagen 5.14 Pez Betta (Morricketín) sin relación alguna con el dataset, en el cuadro de diálogo de *Prueba Rápida*

En el siguiente enlace se pueden encontrar URLs a las categorías de los peces y/o mariscos, 29 URLs en total; para que dichas rutas se puedan ingresar en la caja de texto para realizar una prueba o comprobación rápida:

https://github.com/IsaacIsaias/seafood-classifiers/blob/main/seafood_URLs.txt

5.2.2.2 Azure Cloud Shell

El archivo también posee un código que se puede ejecutar en el Bash de Azure Cloud Shell, pues de tal forma se pueda realizar una llamada al punto de conexión de predicción de un modelo a través de HTTP.

Para realizarlo a través del Bash de Cloud Shell, se accede al Portal de Azure (<https://portal.azure.com>) y se inicia sesión.

Después en la parte superior del Portal, en los controles globales, se selecciona Cloud Shell.

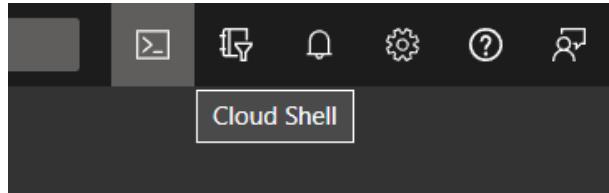


Imagen 5.15 Selección de Cloud Shell

Después se abre Azure Cloud Shell, sin embargo al no tener ningún almacenamiento montado, no es posible iniciararlo, por lo que solo basta en seleccionar *Crear almacenamiento*.

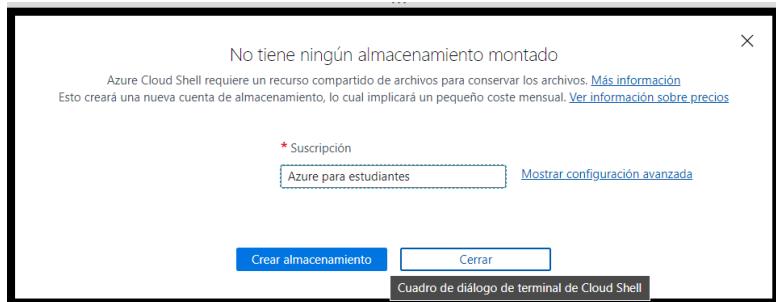
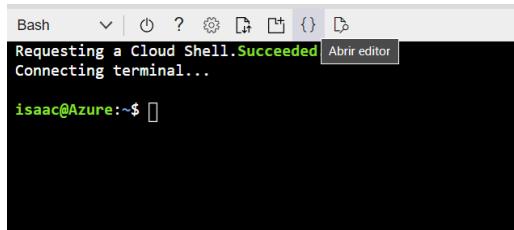


Imagen 5.16 Selección de Crear almacenamiento

Después, se ejecuta la consola; y sugiero abrir el editor para que sea más simple escribir el código y evitar posibles errores.

Imagen 5.17 Selección *Abrir editor* en la CLI

Al abrir el editor de texto, se copia el código que se encuentra dentro del enlace siguiente:

https://github.com/IsaacIsaias/seafood-classifiers/blob/main/seafood_URLs.txt

Este código no se cambia para nada, más que la *x* que se encuentra en la tercera línea. La *x* siempre se reemplaza por la URL de la imagen que se desea predecir.

```
Untitled •
1 curl https://southcentralus.api.cognitive.microsoft.com/customvision/v3.0/Prediction/5e1f4dc2-9c11-460b-9c85-ab
2 > -H "Content-Type: application/json" \
3 > -d "{\"url\" : 'https://previews.123rf.com/images/nutria3000/nutria30001908/nutria3000190800210/128725826-grill
4 > | jq '.'
```

Imagen 5.18 Código escrito en el editor de la CLI

Después de tener el código escrito, se copia todo el código y se va a pegar en el *Bash* con el clic derecho y después el comando *Pegar*.

```
isaac@Azure:~$ curl https://southcentralus.api.cognitive.microsoft.com/customvision/v3.0/Prediction/5e1f4dc2-9c11-460b-9c85-abc65d3c7b00/classify/iterations/Model1201?url \
> -H "Prediction-Key: ffb0db5ced4212aa13c3ceba476623" \
> -H "Content-Type: application/json" \
> -d "{\"url\" : 'https://previews.123rf.com/images/nutria3000/nutria30001908/nutria3000190800210/128725826-grilled-horse-mackerel.jpg'}" \
> | jq '.'
% Total    % Received % Xferd  Average Speed   Time     Time      Current
               Dload  Upload   Total   Spent    Left  Speed
100 1217  100 1089  100  128   377   44  0:00:02  0:00:02 --:--:-- 422
f
```

Imagen 5.19 Código en el Bash de Cloud Shell

Finalmente, se obtiene un archivo JSON con el *id* de la imagen, el id del *proyecto*, la *iteración*, la fecha y hora de *creación* y las respectivas probabilidades de las 9 categorías de los mariscos. La predicción para el pez *Hourse Mackerel* con el siguiente URL es:

<https://previews.123rf.com/images/nutria3000/nutria30001908/nutria3000190800210/128725826-grilled-horse-mackerel.jpg>



Imagen 5.20 Marisco a analizar a través de Cloud Shell

Finalmente obtenemos como resultado el siguiente JSON; en otras palabras, simplemente un diccionario de Python.

```
{
  "id": "2970bf37-946c-4a75-8c33-50ddf77d8af1",
  "project": "5e1f4dc2-9c11-460b-9c85-abc65d3c7b00",
  "iteration": "b7bc461c-efb3-4fc5-a59b-245fed44ceaf",
  "created": "2021-11-13T03:44:55.091Z",
  "predictions": [
    {
      "probability": 0.31042862,
      "tagId": "bda54173-cda5-4919-9358-a4aa9cf108ed",
      "tagName": "Hourse Mackerel"
    },
    {
      "probability": 0.24222657,
      "tagId": "847f3178-2dd6-43df-9f2e-827ad95bd799",
      "tagName": "Striped Red Mullet"
    },
    {
      "probability": 0.19174922,
      "tagId": "d12b0449-a760-4a61-a0af-e1037c6d2b41",
      "tagName": "Gilt-Head Bream"
    },
    {
      "probability": 0.0867266,
      "tagId": "e78842e0-417b-4a46-aedb-c38fb7d31b7",
      "tagName": "Red Sea Bream"
    },
    {
      "probability": 0.053655013,
      "tagId": "0e98a7b9-ce8d-4e8d-a82f-9d13ae5646bd",
      "tagName": "Trout"
    },
    {
      "probability": 0.03769728,
      "tagId": "9904e22b-ed2e-4deb-ad6d-0965d6b15312",
      "tagName": "Black Sea Sprat"
    },
    {
      "probability": 0.031733055,
      "tagId": "0e7d98ad-c107-44e1-a516-aaa55bf9a84c",
      "tagName": "Red Mullet"
    },
    {
      "probability": 0.031675413,
      "tagId": "5375de8f-52b6-4e96-9628-0f54e56e3c3e",
      "tagName": "Shrimp"
    },
    {
      "probability": 0.014108225,
      "tagId": "28977206-1118-498b-9461-4afea56c2bf6",
      "tagName": "Sea Bass"
    }
  ]
}
```

Imagen 5.21 Probabilidades por categoría para la

Imagen

Según las predicciones del modelo, el marisco es un *Hourse Mackerel*, por lo que la prueba ha sido correcta con una predicción del 31.04%.

5.3 Diferencias entre los Modelos

Entre los dos clasificadores que se realizaron, las diferencias más notorias son:

- Se necesita experiencia, tiempo y conocimientos teóricos/prácticos para implementar de maneara adecuada un clasificador CNN, lo que puede consumir tiempo para el programador.
- Es muy simple y rápido desarrollar un clasificador con la API de Custom Vision sin necesidad de ser un programador pues no se utiliza código, sin embargo Azure puede cobrarte si se desean ciertas propiedades en el modelo, como un tiempo de entrenamiento muy largo.

Ambos clasificadores son muy buenos porque son muy precisos, pero en el clasificador CNN se le puede modificar al código para mejorarlo, sin embargo en el clasificador con Custom Vision no se sabe cómo está construido el modelo, solo se obtienen entradas y salidas.

5.4 Guardar un Notebook en GitHub

Para guardar el Notebook en el repositorio público, solamente se selecciona *Guardar una copia en GitHub*, del menú *Archivo*.

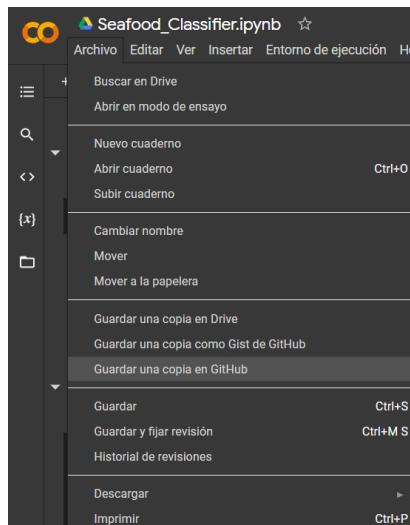


Imagen 5.22 Guardar una copia del Notebook en GitHub

En el cuadro que aparece a continuación, se selecciona el repositorio, la rama y el mensaje de confirmación para el commit. Después se da en *Aceptar*.

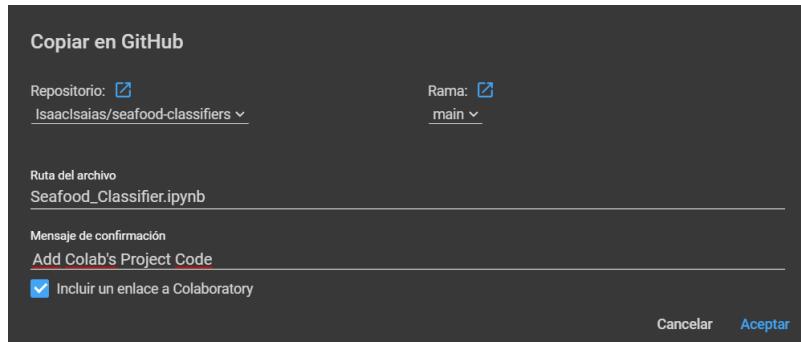


Imagen 5.23 Guardado del código del proyecto en GitHub

Al entrar al repositorio se observa que el archivo *Seafood_Classifier.ipynb* tiene un minuto que se le hizo una modificación, por lo que el commit al repositorio fue correcto.

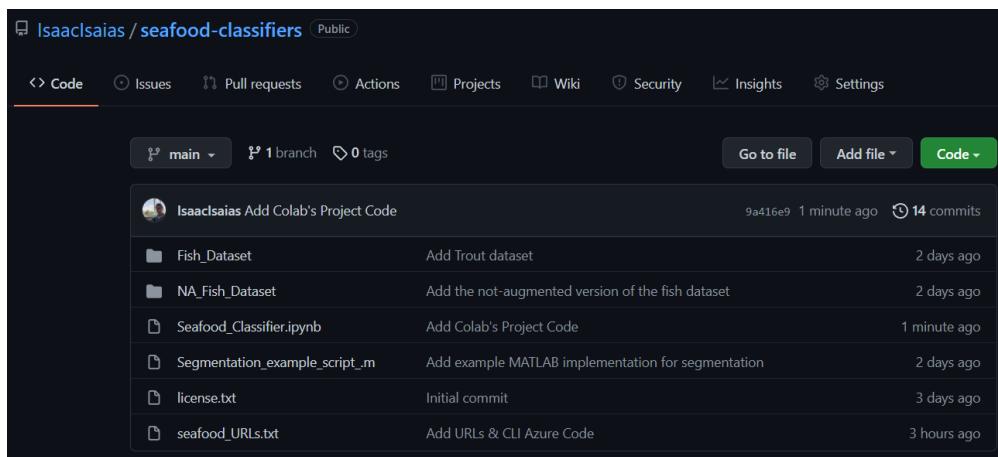


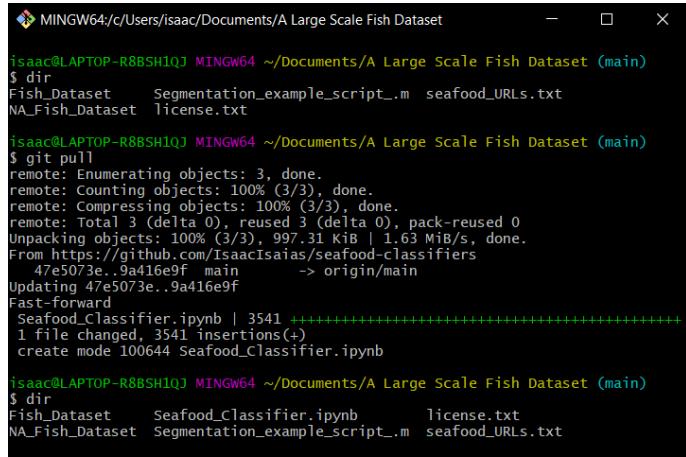
Imagen 5.24 Guardado del código del proyecto en GitHub

Al parecer ya está todo hecho, sin embargo aún falta una última acción. Esta acción es la de traer el Notebook del repositorio remoto al repositorio local, para que los mismos archivos estén en ambos repositorios.

Al ubicarse en el directorio central del repositorio y ejecutar el comando *dir*, se muestran los archivos disponibles, pero el Notebook no se encuentra en el directorio.

Para traer el Notebook, simplemente se ejecuta el comando *git pull*.

Al ejecutar nuevamente el comando *dir* ya se puede apreciar que en el directorio central del repositorio local aparece un archivo extra que previamente no se encontraba allí, siendo el archivo nuevo el Notebook titulado *Seafood_Classifier.ipynb* .



```
MINGW64:/c/Users/isaac/Documents/A Large Scale Fish Dataset
isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ dir
Fish_Dataset Segmentation_example_script_.m seafood_URLs.txt
NA_Fish_Dataset license.txt

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 997.31 KiB | 1.63 MiB/s, done.
From https://github.com/IsaacIsaias/seafood-classifiers
  47e5073e..9a416e9f main      -> origin/main
Updating 47e5073e..9a416e9f
Fast-forward
 Seafood_Classifier.ipynb | 3541 ++++++
 1 file changed, 3541 insertions(+)
 create mode 100644 Seafood_Classifier.ipynb

isaac@LAPTOP-R8BSH1QJ MINGW64 ~/Documents/A Large Scale Fish Dataset (main)
$ dir
Fish_Dataset Segmentation_example_script_.m seafood_URLs.txt
NA_Fish_Dataset Seafood_Classifier.ipynb license.txt
```

Imagen 6.25 Confirmación del *pull* al repositorio local

CONCLUSIONES

El finalizar este proyecto me deja un buen sabor de boca, más que nada por el gran enriquecimiento obtuve al final de este; pues tuve que dominar herramientas para su elaboración y de igual manera, aprender una gran cantidad de conocimientos teóricos y prácticos a lo largo de este tiempo de trabajo.

Puedo decir que afortunadamente se lograron los objetivos y alcances previstos para este proyecto, pues se esperaba obtener como mínimo en el clasificador del modelo CNN una precisión superior al 90% y la precisión obtenida fue del 95%. De igual manera, el clasificador No-Code también tuvo un buen desempeño al predecir correctamente un pez frito de esa categoría. Por último, todo el manejo de los repositorios a través de Git Bash fue satisfactorio, pues no se tuvieron problemas más que algunos ligeros inconvenientes para cargar todos los datos del dataset, pero solo con cambiar el `http,postBuffer` de Git se solucionó.

Como se pudo apreciar, el proyecto tiene muy buenos resultados y estoy satisfecho por ello. Sin embargo, existen áreas de mejora, algunas de estas áreas son:

- Intentar probar el modelo CNN con platillos de dichos mariscos u otras imágenes de las mismas categorías de estos.
- Mejorar la precisión del modelo CNN modificando la arquitectura del modelo, por ejemplo, con la adición de capas como *Dropout* o de *Batch Normalization*.
- Realizar un clasificador usando Redes Neuronales Convolucionales Siamesas, una estructura comparativa en la que se analiza diferencias entre imágenes.
- Exportar el modelo No-Code para ser usado en dispositivos. Se puede exportar en TensorFlow, TensorFlowJS, CoreML, ONNX y Docker; para dispositivos Android, iOS o con arquitectura de Windows, Linux o ARM.

REFERENCIAS

- Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. Nueva York: Springer.
- Ahmad, I. (2020). *40 Algorithms Every Programmer Should Know: Hone your problem-solving skills by learning different algorithms and their implementation in Python*. Birmingham: Packt Publishing.
- Apache. (2015). *MXNet*. Obtenido de <https://github.com/apache/incubator-mxnet>
- Bell, P., & Beer, B. (2015). *Introducing GitHub: A Non-Technical Guide*. Sebastopol, California: O'Reilly Media Inc.
- Berkeley Vision and Learning Center (BVLC). (2013). *Caffe*. Obtenido de <https://github.com/BVLC/caffe>
- Borra, S., Thanki, R., & Dey, N. (2019). *Satellite Image Analysis: Clustering and Classification*. Singapur: Springer.
- Chan, Y.-H. (s.f.). *Department of Electronic and Information Engineering*. Obtenido de The Hong Kong Polytechnic University: <http://www.eie.polyu.edu.hk/~enyhchan/images.pdf>
- Chollet, F. (2015). *Keras*. Obtenido de <https://github.com/keras-team/keras>
- Cowley, J. (7 de Diciembre de 2018). *Redes Neuronales Convolucionales*. Obtenido de IBM Development: <https://developer.ibm.com/es/articles/cc-convolutional-neural-network-vision-recognition/>
- Dey, N., Bhatt, C., & Ashour, A. S. (2018). *Big Data for Remote Sensing: Visualization, Analysis and Interpretation*. Cham, Suiza: Springer.
- Dey, S. (2018). *Hands-On Image Processing with Python: Expert techniques for advanced image analysis and effective interpretation of image data*. Birmingham: Packt Publishing.

- Dey, S. (2020). *Python Image Processing Cookbook: Over 60 recipes to help you perform complex image processing and computer vision tasks with ease*. Birmingham: Packt Publishing.
- Efford, N. (2000). *Digital Image Processing; a practical introduction using JAVA*. Pearson Education.
- Event Horizon Telescope (EHT). (2019). *Black Hole Image Makes History*. Obtenido de Jet Propulsion Laboratory: <https://d2pn8kiwq2w21t.cloudfront.net/images/imagesuniverse20190410blackhole20190410-16.width-1320.jpg>
- Facebook AI Research. (2016). *PyTorch*. Obtenido de <https://github.com/pytorch/pytorch>
- Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, XV*, 315-323.
- González, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4ta ed.). Harlow: Pearson Education.
- Google Brain. (2015). *TensorFlow*. Obtenido de <https://github.com/tensorflow/tensorflow>
- Google LLC. (s.f.). *Google Colaboratory*. Obtenido de <https://colab.research.google.com/>
- Gorner, M. (2021). *TensorFlow, Keras and deep learning, without a PhD*. Obtenido de Google Developers Codelabs: <https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/index.html#10>
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 770-778. Obtenido de <https://arxiv.org/pdf/1512.03385.pdf>
- IBM. (03 de Junio de 2020). *Artificial Intelligence*. Obtenido de IBM Cloud Education: <https://www.ibm.com/cloud/learn/what-is-artificial-intelligence>

- IBM. (2021). *SDK vs API: What's the Difference?* Obtenido de IBM Cloud Education: <https://www.ibm.com/cloud/blog/sdk-vs-api>
- Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv*. Obtenido de <https://arxiv.org/pdf/1502.03167.pdf>
- Kaggle. (s.f.). *Kaggle*. Obtenido de <https://www.kaggle.com/>
- Kar, K. (2020). *Mastering Computer Vision with Tensorflow 2.x: Build advanced computer vision applications using machine learning and deep learning techniques*. Birmingham: Packt Publishing.
- Karbhari, V. M., & Ansari, F. (2009). *Structural health monitoring of civil infrastructure systems*. Cornwall: CRC Press.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neral Networks. *Communications of the ACM*, LX, 84-90.
- Kumar, A., Upadhyay, P., & Kumar, A. S. (2020). *Fuzzy Machine Learning Algorithms for Remote Sensing Image Classification*. Boca Ratón, Florida: CRC Press.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 436-444.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 2278-2324.
- Leonard, M., Serrano, L., Camacho, C., Cook, A., Staab, J., Carell, S., . . . Jiang, D. (s.f.). *Deep Learning Nanodegree program*. Obtenido de Udacity: <https://www.udacity.com/course/deep-learning-nanodegree--nd101>
- Lin, M., Chen, Q., & Yan, S. (16 de Diciembre de 2013). Network In Network. *Computing Research Repository*. Obtenido de arXiv.org: <https://arxiv.org/pdf/1312.4400.pdf>
- Loeliger, J., & McCullough, M. (2012). *Version Control with Git*. Sebastopol, California: O'Reilly Media Inc.

- matplotlib. (s.f.). *matplotlib*. Obtenido de <https://github.com/matplotlib/matplotlib>
- McCarthy, J. (12 de Noviembre de 2007). *What is Artificial Intelligence?* Obtenido de John McCarthy's Home Page: <http://www-formal.stanford.edu/jmc/whatisai.pdf>
- McCarthy, J., Minsky, M., Rochester, N., & Shannon, C. (1956). *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*. Hanover, New Hampshire.
- Microsoft Corporation. (s.f.). *Azure Marketplace*. Obtenido de <https://azuremarketplace.microsoft.com/es/>
- Microsoft Corporation. (s.f.). *Custom Vision*. Obtenido de <https://www.customvision.ai/>
- Microsoft Corporation. (s.f.). *GitHub*. Obtenido de <https://github.com/>
- Microsoft Corporation. (s.f.). *Introducción a los aspectos básicos de Azure*. Obtenido de Microsoft Learn: <https://docs.microsoft.com/es-es/learn/modules/intro-to-azure-fundamentals/>
- Microsoft Corporation. (s.f.). *Microsoft Cognitive Services*. Obtenido de <https://azure.microsoft.com/en-us/services/cognitive-services/>
- Microsoft Research. (2016). *Microsoft Cognitive Toolkit (CNTK)*. Obtenido de <https://github.com/Microsoft/CNTK>
- MIT CSAIL. (2021). *Mission & History*. Obtenido de MIT Computer Science & Artificial Intelligence Lab: <https://www.csail.mit.edu/about/mission-history>
- Montreal Institute for Learning Algorithms (MILA). (2007). *Theano*. Obtenido de <https://github.com/Theano/Theano>
- Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*.
- NumPy. (s.f.). *What is NumPy?* Obtenido de Numpy: <https://github.com/numpy/numpy>
- OpenCV. (s.f.). *OpenCV*. Obtenido de <https://github.com/opencv/opencv>

Patel, A. A. (2019). *Hands-On Unsupervised Learning Using Python*. California: O'Reilly Media, Inc.

Pérez, E. I. (s.f.). *Curso Profesional de Python*. Obtenido de CódigoFacilito: <https://codigofacilito.com/cursos/python-profesional>

Preferred Networks Inc. (2015). *Chainer*. Obtenido de <https://github.com/chainer/chainer>

Project Jupyter. (s.f.). *Jupyter*. Obtenido de <https://jupyter.org/>

PyPI. (s.f.). Obtenido de Python Package Index: <https://pypi.org/>

Python. (s.f.). *glob*. Obtenido de glob - Unix style pathname pattern expansion: <https://docs.python.org/3/library/glob.html>

Python. (s.f.). *os*. Obtenido de os - Miscellaneous operating system interfaces: <https://docs.python.org/3/library/os.html#module-os>

Python. (s.f.). *os.path*. Obtenido de os.path - Common pathname manipulations: <https://docs.python.org/3/library/os.path.html#module-os.path>

Python. (s.f.). *pathlib*. Obtenido de pathlib - Object-oriented filesystem paths: <https://docs.python.org/3/library/pathlib.html#module-pathlib>

Python. (s.f.). *What is Python? Executive Summary*. Obtenido de Python.org: <https://www.python.org/doc/essays/blurb/>

PyTorch. (s.f.). *Ecosystem Tools*. Obtenido de <https://pytorch.org/ecosystem/>

Ramachandran, P., Zoph, B., & Le, Q. V. (27 de Octubre de 2017). Searching for activation function. *arXiv: Neural and Evolutionary Computing*. Obtenido de arXiv.org: <https://arxiv.org/pdf/1710.05941.pdf>

Red Hat. (2020). *What is a REST API?* Obtenido de Red Hat: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

Rivas, P. (2020). *Deep Learning for Begineers*. Birmingham: Packt Publishing.

- Russakovsky, O., & Deng, J. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 211-252.
 doi:<https://doi.org/10.1007/s11263-015-0816-y>
- Russell, S. J., & Norvig, P. (2021). *Artificial Intelligence. A Modern Approach* (Cuarta ed.). Harlow: Pearson Education.
- scikit-learn. (s.f.). *scikit-learn*. Obtenido de <https://github.com/scikit-learn/scikit-learn>
- Shanmugamani, R. (2018). *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using Tensorflow and Keras*. Birmingham: Packt Publishing.
- Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recogniton. *Computing Research Repository*. Obtenido de https://www.robots.ox.ac.uk/~vgg/research/very_deep/
- Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Visual Recognition*. Obtenido de Visual Geometry Group. University of Oxford: https://www.robots.ox.ac.uk/~vgg/research/very_deep/
- Skymind Engineering Team. (2014). *Deeplearning4j*. Obtenido de <https://github.com/deeplearning4j/deeplearning4j>
- SoloLearn. (s.f.). *Python Core*. Obtenido de SoloLearn: <https://www.sololearn.com/learning/1073>
- Stanford University. (Spring de 2021). *CS231n: Convolutional Neural Networks for Visual Recognition*. Obtenido de CS231n Home: <https://cs231n.github.io/convolutional-networks/>
- Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2017). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 4278-4284. Obtenido de <https://arxiv.org/pdf/1602.07261.pdf>

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going Deeper with Convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition*. Obtenido de <https://arxiv.org/pdf/1409.4842.pdf>

The pandas development team. (s.f.). *pandas*. Obtenido de pandas-dev: <https://github.com/pandas-dev/pandas>

Torvalds, L. (2005). *Git*. Obtenido de <http://git-scm.com/>

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind, LIX*, 433-460. doi:<https://doi.org/10.1093/mind/LIX.236.433>

Waskom, M. (s.f.). *seaborn*. Obtenido de <https://github.com/mwaskom/seaborn>

Wozniewicz, B. (2019). *The Difference Between a Framework and a Library*. Obtenido de freeCodeCamp: <https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/>

Yalçın, O. G. (2020). *Applied Neural Networks with TensorFlow 2: API Oriented Deep Learning with Python*. Estambul: Apress.

Zafar, I., Tzanidou, G., Burton, R., Patel, N., & Araujo, L. (2018). *Hands-On Convolutional Neural Networks with Tensorflow: Solve computer vision problems with modeling in TensorFlow and Python*. Birmingham: Packt Publishing.

Zeiler, M. D., & Fergus, R. (28 de Noviembre de 2014). Visualizing and Understanding Convolutional Networks. *European Conference on Computer Vision*. Obtenido de arXiv.org: <https://arxiv.org/pdf/1311.2901.pdf>