UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Detección de bajas de aves

Presentado por Saúl Vetia de Juana
en Universidad de Burgos — 28 de enero de 2017
Tutor: Carlos Cambra Baseca
Roberto Carlos Casado Vara

Índice general

Índice de figuras

Apéndice A

Plan de proyecto software

A.1. Introducción

El proceso de realización de este trabajo se puede clasificar en tres etapas: “Comprensión del código”, “investigación teórica” y “realización de pruebas”.

• Durante la etapa de comprensión del código el objetivo fue entender y adaptar de la forma más básica el clasificador de ejemplo proporcionado, de manera que pudiera leer las imágenes del proyecto, con el objetivo de ser capaz de identificar los distintos bloques del programa y entender su función.

• Durante la etapa de investigación teórica se realizó un proceso de investigación y documentación de todos los aspectos relevantes sobre redes neuronales, tanto para aportar un marco teórico que permitiese obtener un conocimiento general sobre la materia como para poder entender mejor el código y cómo abordarlo.

• Durante la etapa de realización de pruebas se plantearon múltiples métodos para abordar el problema y se estudió su utilidad. Esta etapa comprende tanto la búsqueda de las herramientas necesarias para ello (funciones y comandos) como su implementación y comprobación de resultados.

A.2. Planificación temporal

Sprint 1

Duración: 2 semanas.
Horas de trabajo estimadas: 30

Durante este sprint se realizó todo el proceso de comprensión del código de ejemplo y adaptación del mismo. Se realizaron diversas ejecuciones para analizar su funcionamiento y se creó la primera versión funcional del programa, en la que simplemente se cambian las rutas de las imágenes y se eliminan los fragmentos de código innecesarios.

Eje X: Días
Eje Y: Horas de trabajo totales

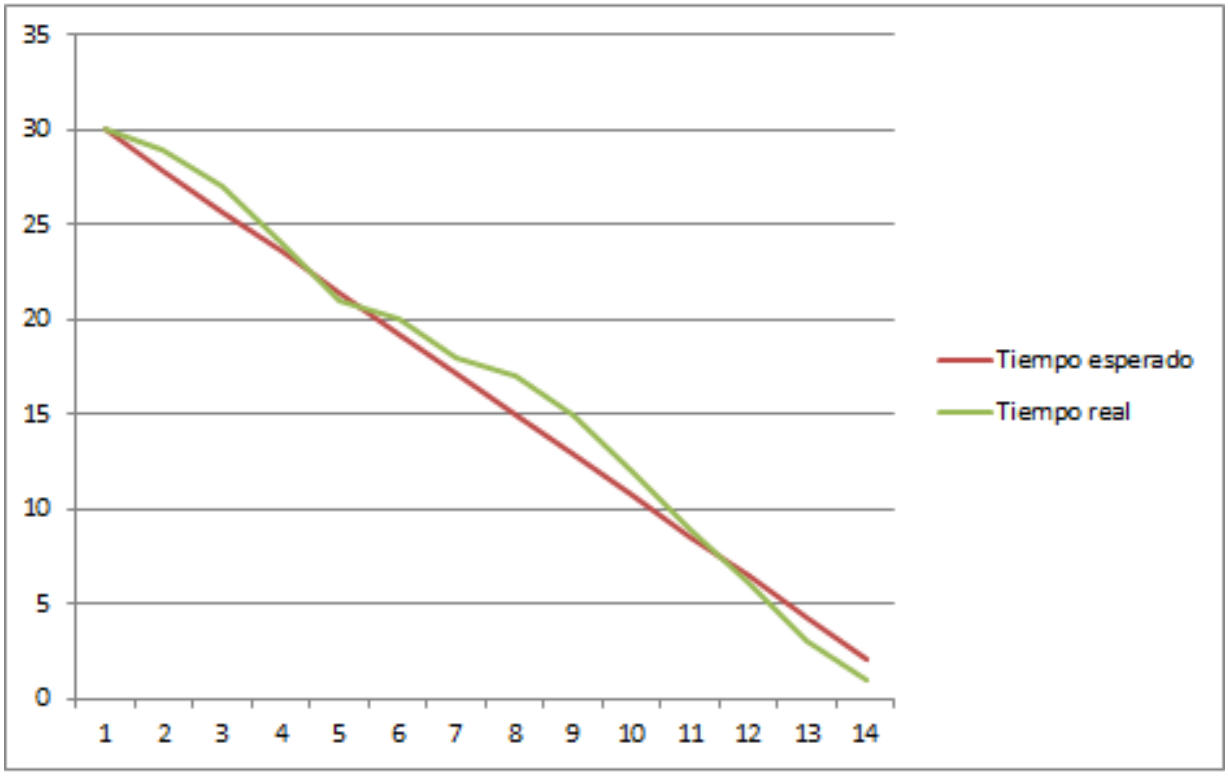

Ilustración 1: Burndown chart 1

Sprint 2

Duración: 2 semanas.
Horas de trabajo estimadas: 40

Durante este sprint se realizó la parte de investigación teórica, así como se completaron sus respectivos apartados en la memoria.

Eje X: Días
Eje Y: Horas de trabajo totales

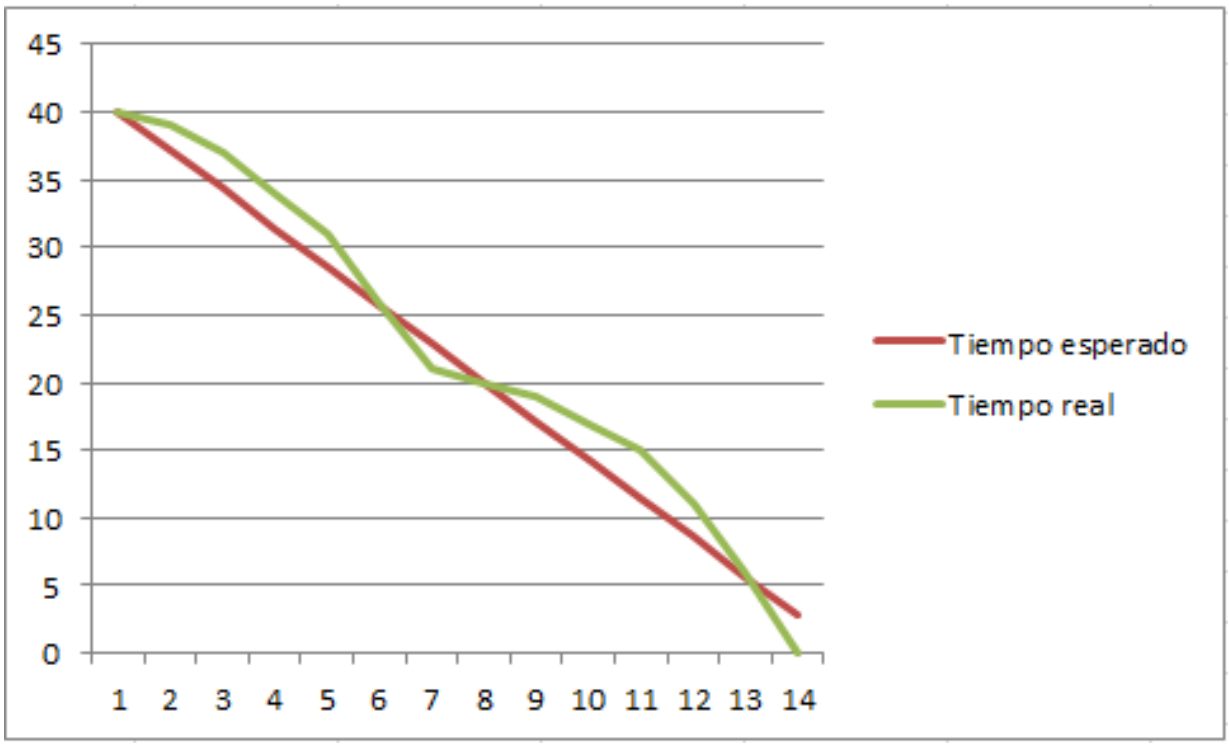

Ilustración 2: Burndown chart 2

Sprint 3

Duración: 31 días
Horas de trabajo estimadas: 70

Durante este sprint se realizaron todas las pruebas con el código y se completaron algunos apartados de la memoria.

Eje X: Días
Eje Y: Horas de trabajo totales

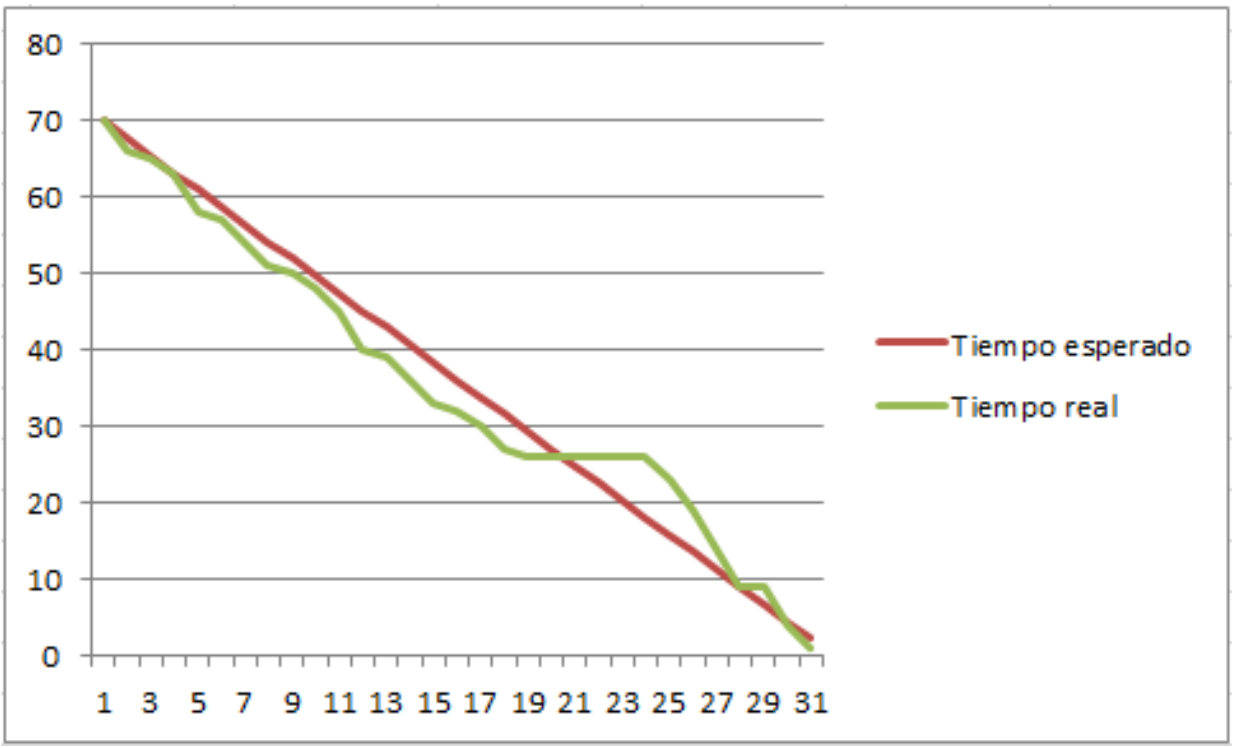

Ilustración 3: Burndown chart 3

Sprint 4

Duración estimada: 1 semana
Horas de trabajo estimadas: 30

Durante este sprint se terminó el proyecto. Esto incluye la finalización de la memoria y las últimas modificaciones del código.

Eje X: Días
Eje Y: Horas de trabajo totales

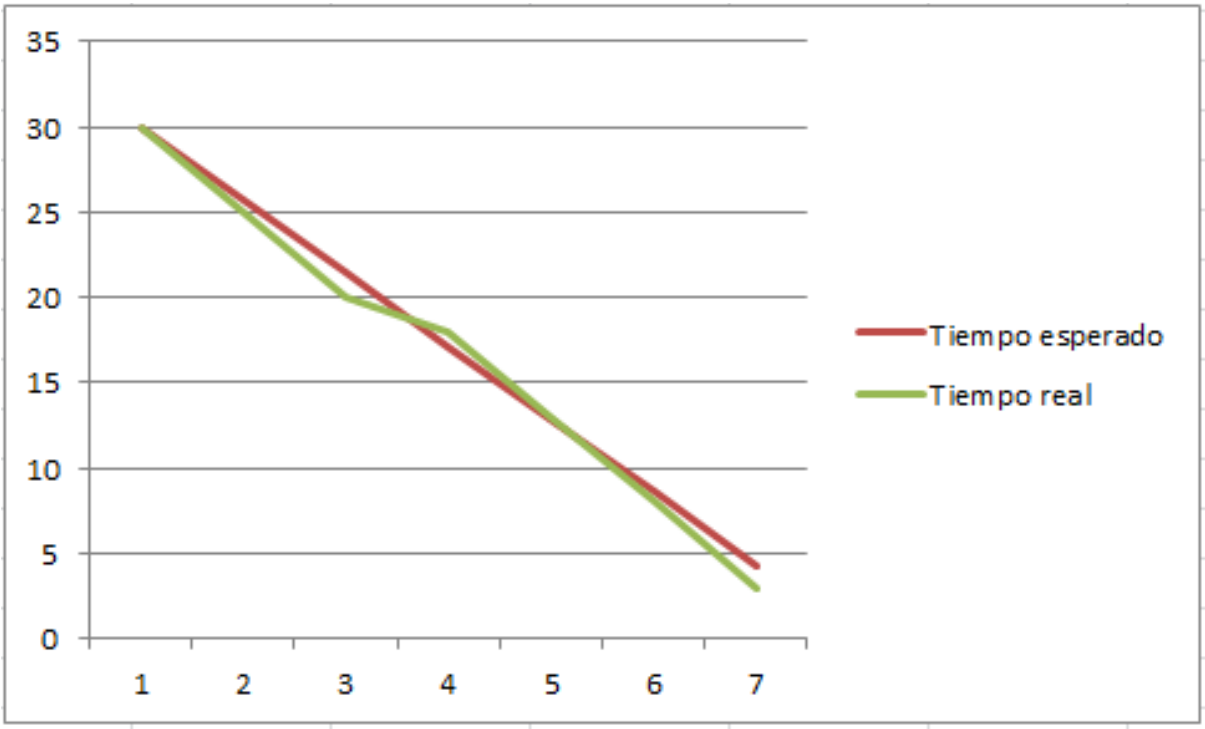

Ilustración 4: Burndown chart 4

Apéndice B

Especificación de requisitos

B.1. Introducción.

Este anexo contiene la especificación de requisitos con los que debe cumplir el sistema desarrollado.

B.2. Objetivos generales.

El proyecto persigue los siguientes objetivos:

-Entrenar una red neuronal convolucional capaz de clasificar fotografías de gallinas en dos grupos: vivas y muertas.

-Facilitar la comprensión de los resultados gráficamente.

-Predecir, una vez entrenada la red, cualquier fotografía que se le pase al programa.

B.3. Catálogo de requisitos.

A continuación se definen los requisitos funcionales y no funcionales del proyecto.

Requisitos funcionales

•RF1-Entrenar una red neuronal convolucional que prediga adecuadamente las instancias de entrenamiento y validación.

- •RF1.1-Definir el modelo de red utilizando una estructura apropiada, de acuerdo a lo estudiado en el marco teórico.
- •RF1.2-Entrenar el modelo aplicando las técnicas necesarias para evitar sobreajuste.

•RF2-Contrastar los resultados obtenidos mediante su visualización gráfica, para facilitar su comprensión.

•RF3-Clasificar imagen: Se proporciona una fotografía de una jaula al programa y se clasifica.

Requisitos no funcionales

•RNF1-Rendimiento: El consumo de recursos computacionales al entrenar la red debe ser bajo.

Apéndice C

Especificación de diseño

C.1. Introducción.

En este apartado se explica el tipo de datos con el que trabaja el programa y el orden de ejecución de los componentes del mismo.

C.2. Diseño de datos.

Los datos de entrada que maneja el programa son las imágenes de las jaulas con gallinas. Se trata de imágenes JPG de 4000*3000 píxeles.

Los datos de salida se corresponden con los resultados ofrecidos por la red neuronal, es decir, los valores de precisión y pérdida correspondientes a los conjuntos de entrenamiento y validación.

•**Precisión**: Medida que clasifica de forma porcentual el número de instancias que la red clasifica correctamente. Una precisión de 0'7 significa que predice correctamente el 70% de las instancias. Incluye los parámetros *accuracy* y *val_accuracy*

•**Pérdida**: Es el número de errores cometidos para cada muestra en los conjuntos de entrenamiento y validación. Incluye los parámetros *loss* y *val_loss*.

C.3. Diseño procedimental.

Al tratarse de un notebook de Jupyter, las celdas están dispuestas en el orden de ejecución esperado.

```
1  import urllib.request
2  import os
3  import zipfile
4
5  import matplotlib.image as mpimg
6  import matplotlib.pyplot as plt
7
8
9  import tensorflow as tf
10
11 from tensorflow.keras.optimizers import RMSprop
12 from tensorflow.keras.preprocessing.image import ImageDataGenerator
13 from tensorflow.keras.preprocessing import image
14
15
16 import numpy as np
```

Ilustración 5: Librerias

Ubicación de imágenes de modelo y de test.

```
#Directorio base
base_dir = 'C:/Users/Saul/Desktop/SAUL/INFORMATICA/UBU 4º/2º semestre/TFG/Fotos_Proyecto_Detección_de_Bajas'

train_dir = os.path.join(base_dir, 'Entrenamiento')
validation_dir = os.path.join(base_dir, 'Validacion')

# Directorio con las fotos de gallinas clasificadas
train_alive_dir = os.path.join(train_dir, 'Gallinas_vivas')
train_dead_dir = os.path.join(train_dir, 'Gallinas_muertas')

# Directorio con las imágenes de modelo
validation_alive_dir = os.path.join(validation_dir, 'Gallinas_vivas')
validation_dead_dir = os.path.join(validation_dir, 'Gallinas_muertas')
```

Ilustración 6: Definir las rutas de las imágenes

Versión original

```
model = tf.keras.models.Sequential([
    # tf.keras.layers.Conv2D(filtros, tamaño_kernel, activacion, input_shape)
    # input_size es la forma deseada de la imagen (150x150) con 3 bytes de color
    # Bloque de convolución 1
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 2
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Bloque de convolución 3
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # "Aplana" los resultados para pasárselos a una DNN (Deep neural network)
    tf.keras.layers.Flatten(),

    # Neuronas de capa oculta: 512
    tf.keras.layers.Dense(512, activation='relu')
    ,
    # Neurona de salida que devuelve dos valores: 0 para gallinas muertas y 1 para gallinas vivas.
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ilustración 7: Definición del modelo

```
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics = ['accuracy'])
```

Ilustración 8: Compilar el modelo

```
# Reescalado de imágenes a 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

# --------------------------------------------------------------------
# Entrenamiento de las imágenes en lotes de 5 utilizando "train_datagen"
# --------------------------------------------------------------------
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=5,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

# --------------------------------------------------------------------
# Validación de las imágenes en lotes de 5 utilizando "test_datagen"
# --------------------------------------------------------------------
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=5,
                                                         class_mode  = 'binary',
                                                         target_size = (150, 150))
```

Ilustración 9: Generación de lotes de tamaño 5

Entrenamiento

```
history = model.fit(train_generator,
                            validation_data=validation_generator,
                            epochs=25,
                            verbose=2)
```

Ilustración 10: Entrenamiento del modelo

```
resultado = model.evaluate(validation_generator, verbose=0)
print(f'Pérdida: {resultado[0]} / Precisión: {resultado[1]}')
```

Ilustración 11: Se muestran los resultados del modelo

```
#------------------------------------------------------------
# Recupero los valores para dibujar las gráficas
#------------------------------------------------------------
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

#------------------------------------------------
# Accuracy vs val_accuracy
#------------------------------------------------
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('accuracy vs val_accuracy')
plt.legend()

plt.figure()

#------------------------------------------------
# Loss vs val_loss
#------------------------------------------------
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('loss vs val_loss')
plt.legend()

plt.show()
```

Ilustración 12: Generación de gráficas

Apéndice D

Documentación técnica de programación

D.1. Introducción.

En este anexo se explican aspectos como la estructura del repositorio del proyecto, el proceso de instalación del entorno de desarrollo y la ejecución de las pruebas realizadas.

D.2. Estructura de directorios.

La distribución de repositorios del proyecto es la siguiente:

URL: https://github.com/svd0009/TFG_Deteccion_Bajas.git

•**/**: Contiene el fichero README y la licencia del proyecto.

•**/docs/:** Contiene la documentación del proyecto.

•**/docs/imgs_mem**/: Contiene las imágenes utilizadas en la memoria.

•**/docs/imgs_app**/: Contiene las imágenes empleadas para entrenar la red.

•**/docs/imgs_app /train**/: Contiene el conjunto de imágenes de entrenamiento.

•**/docs/imgs_app /train/alive**/: Contiene el conjunto de imágenes de entrenamiento de gallinas vivas.

•**/docs/imgs_app /train/dead**/: Contiene el conjunto de imágenes de entrenamiento de gallinas muertas.

•**/docs/imgs_app /test**/: Contiene el conjunto de imágenes de validación.

•**/docs/imgs_app /test/alive**/: Contiene el conjunto de imágenes de validación de gallinas vivas.

•**/docs/imgs_app /test/dead**/: Contiene el conjunto de imágenes de validación de gallinas muertas.

•**/docs/imgs_app /pred**/: Contiene las imágenes utilizadas en las pruebas de predicción.

•**/docs/memo**/: Contiene la memoria y el anexo en formato PDF.

•**/app**/: Contiene los notebook correspondientes a las pruebas y a la versión final.

•**/app/ver**/: Contiene los notebook correspondientes a las distintas pruebas realizadas.

•**/app/fin**/: Contiene la versión final del programa.

D.3. Manual del programador.

En este punto se explicará cómo instalar el entorno de desarrollo de Anaconda3.

Se deben de seguir los siguientes pasos:

1- Descargar el instalador de Anaconda. La última versión disponible es Anaconda3, que incluye Python 3.9. Para ello basta con entrar en la página oficial de Anaconda y descargar el instalador de la última versión.

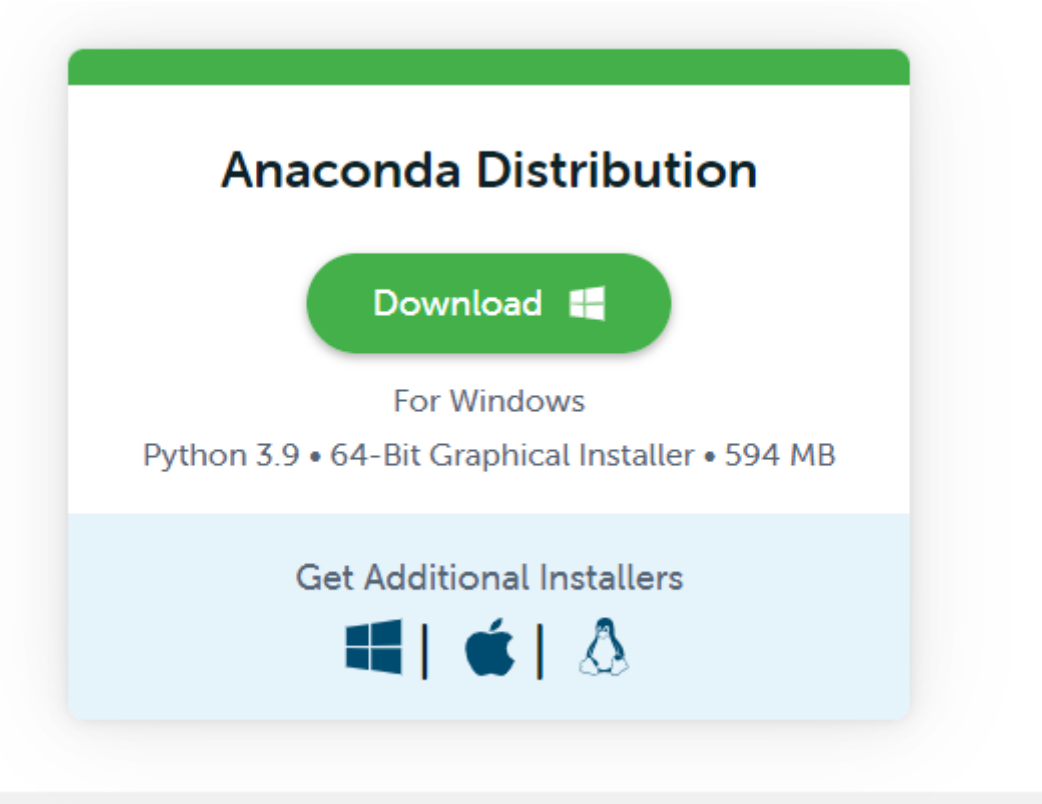

Ilustración 13: Descarga del instalador

2- Una vez descargado el instalador, lo abrimos y seguimos los pasos que se indican

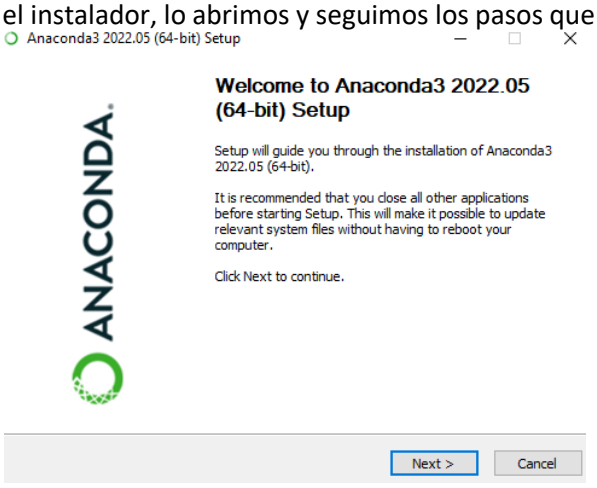

Ilustración 14: Paso 1 del instalador

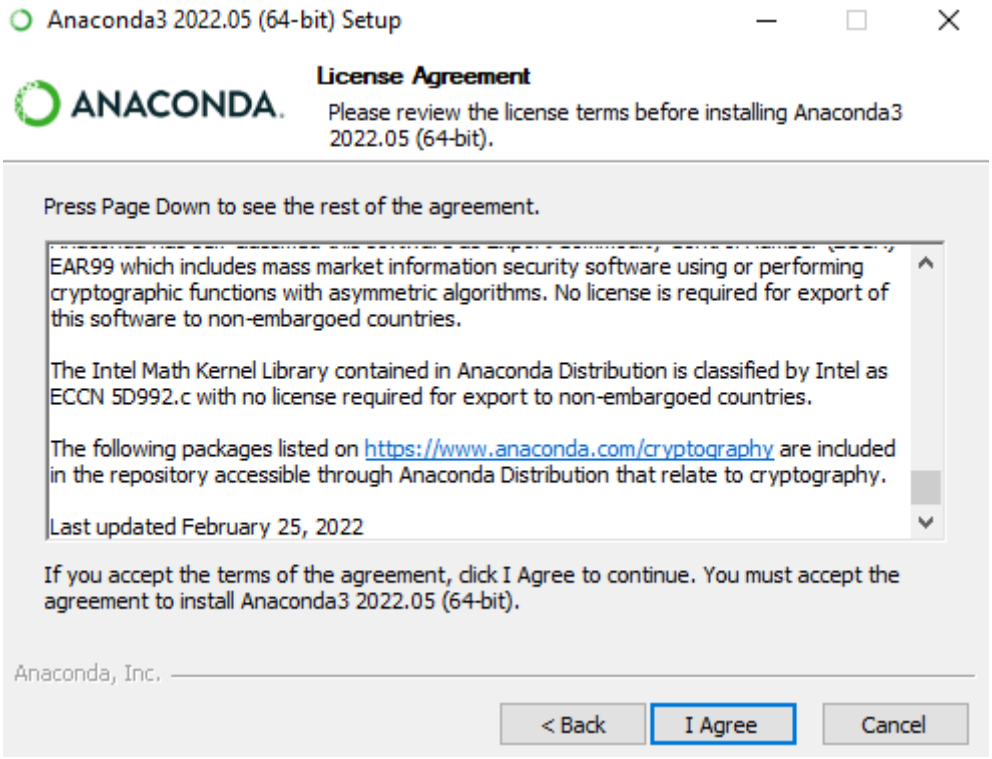

Ilustración 15: Paso 2 del instalador

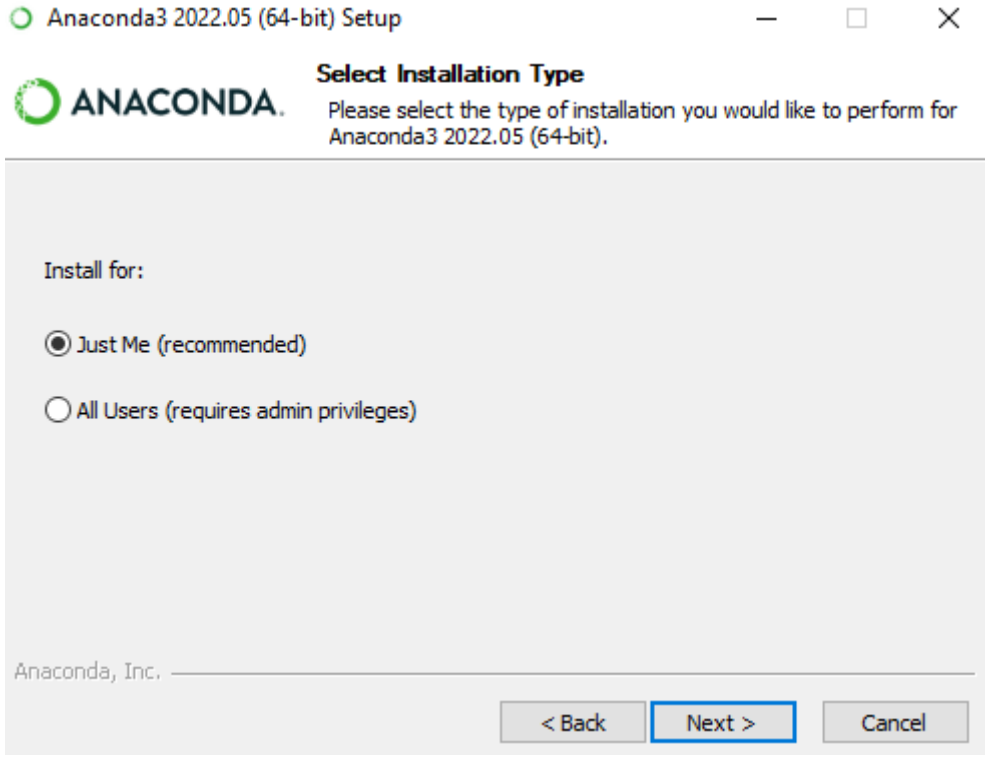

Ilustración 16: Paso 3 del instalador

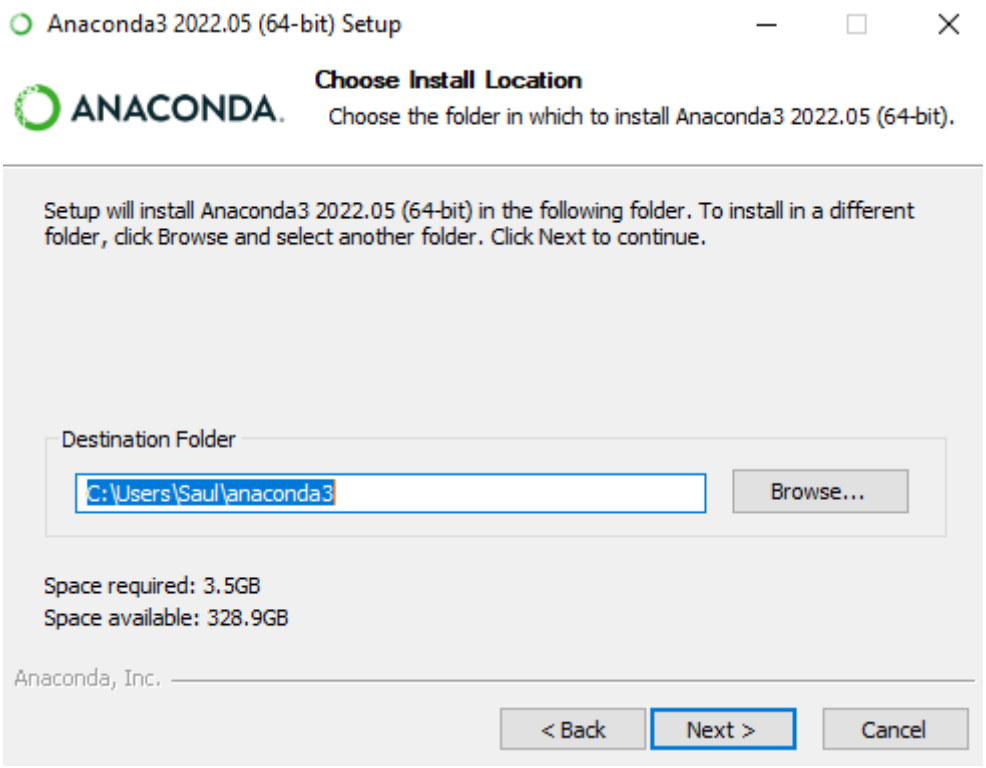

Ilustración 17: Selección de directorio de instalación

Apéndice E

Documentación de usuario

E.1. Introducción.

En este apartado se explicarán las herramientas necesarias para el uso del programa, los pasos necesarios para instalarlas y las modificaciones necesarias para que el programa final funcione en cualquier equipo.

E.2. Requisitos de usuarios.

Requisitos del sistema

• Sistema operativo: Windows 8 o superior, macOS 10.13 o superior, distribuciones Linux Ubuntu, RedHat, CentOS 7 o superior, Mint, o cualquier otra distribución Linux.

• Arquitectura: Windows-64-bit x86, Windows-32-bit x86, MacOS- 64-bit x86 & M1, Linux-64-bit x86, 64-bit aarch64 (AWS Graviton2), 64-bit Power8/Power9, s390x.

• Capacidad: 5GB para descarga e instalación de Anaconda.

Memoria necesaria para el entrenamiento

Para calcular la memoria mínima necesaria para entrenar la red es necesario tener en cuenta los parámetros de la red neuronal. Para ello se calculan los parámetros por cada mapa de características (cada vez que se aplican los filtros a la imagen de entrada) y se multiplica por el número de filtros.
Las imágenes utilizadas utilizan un sistema de codificación de 24 bits (formato JPG).

Cálculo de parámetros

Primera capa de convolución

Tamaño del kernel: 3*3.
Entrada: Imágenes de 150*150 con tres canales (R, G, B).
Tamaño del filtro: 80.

Parámetros por cada mapa de características (feature map): 3*3*3=27.
Parámetros totales: 27*80=2160.

Segunda capa de convolución

Tamaño del kernel: 3*3.
Entrada: Imágenes de 150*150 con tres canales (R, G, B). Recibe los mapas de características de la capa anterior (80).
Tamaño del filtro: 100.

Parámetros por cada mapa de características: 3*3*80=720.
Parámetros totales: 720*100=72000.

Tercera capa de convolución

Tamaño del kernel: 3*3.
Entrada: Imágenes de 150*150 con tres canales (R, G, B). Recibe los mapas de características de la capa anterior (100).
Tamaño del filtro: 100.

Parámetros por cada mapa de características: 3*3*100=900.
Parámetros totales: 900*100=90000.

Cuarta capa de convolución

Tamaño del kernel: 3*3.
Entrada: Imágenes de 150*150 con tres canales (R, G, B). Recibe los mapas de características de la capa anterior (100).
Tamaño del filtro: 100.

Parámetros por cada mapa de características: 3*3*100=900.
Parámetros totales: 900*100=90000.

Total de parámetros

2160+72000+90000+90000=254160.

Cálculo de memoria

Para calcular la memoria requerida por cada capa de convolución se multiplica el tamaño de palabra de las imágenes de entrada por su tamaño en píxeles y por el tamaño del filtro.

Primera capa de convolución

24 bits = 3 bytes

3*150*150*80=5400000 bytes=5’14 MB

Segunda capa de convolución

3*150*150*100=6750000 bytes=6’43 MB

Tercera capa de convolución

3*150*150*100=6750000 bytes=6’43 MB

Cuarta capa de convolución

3*150*150*100=6750000 bytes=6’43 MB

Memoria ocupada por los parámetros de la red

254160*3 Bytes=762480 bytes = 0’72 MB

Total de memoria

5’14+6’43+6’43+6’43 +0’72=25’15 MB

E.3. Instalación.

Para poder utilizar las herramientas de Python necesarias para abrir el proyecto es necesario instalar Anaconda3.

Para ello se aplica el mismo criterio que en el apartado D.3. Manual del programador.