



# Trabajo Final de Máster

---

Redes Neuronales Automodelables

Autor: Carlos Bilbao Lara

Tutor: Alberto Partida Rodríguez



# Índice

01

Introducción

02

Objetivos

03

Marco  
Teórico

04

Marco  
Metodológico

05

Análisis y  
Resultados

06

Conclusiones



# Introducción

Introducción

01

02

Objetivos

03

Marco  
Teórico

04

Marco  
Metodológico

05

Análisis y  
Resultados


06

Conclusiones

# Motivaciones

---

Actualmente la enorme mayoría de proyectos de crear redes neuronales requieren de una fase de probar y mejorar la arquitectura de la red

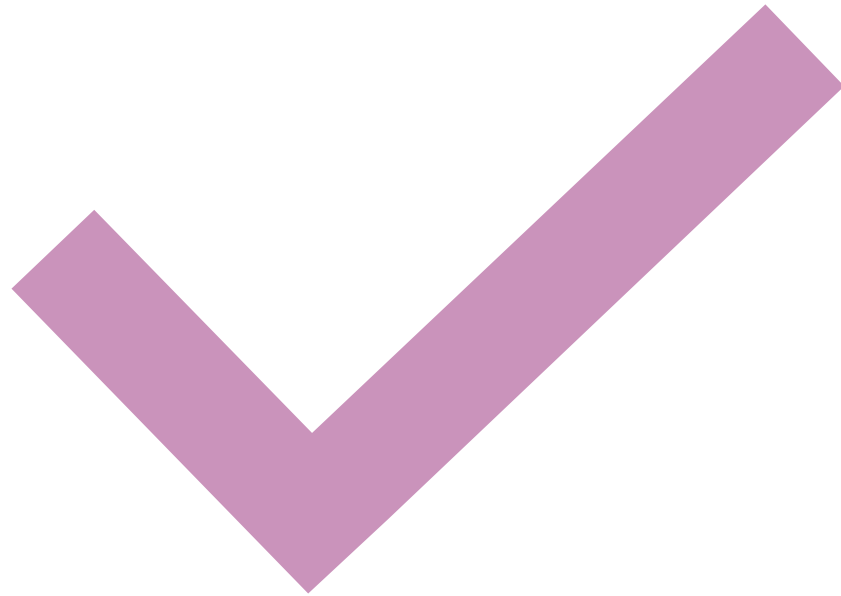


Con el fin de evitar esta tediosa fase, se empieza a automatizar lo máximo posible



Desarrollar un algoritmo evolutivo que sea capaz de, probar y mejorar las redes neuronales

# Objetivos



01  
Introducción

02  
Objetivos

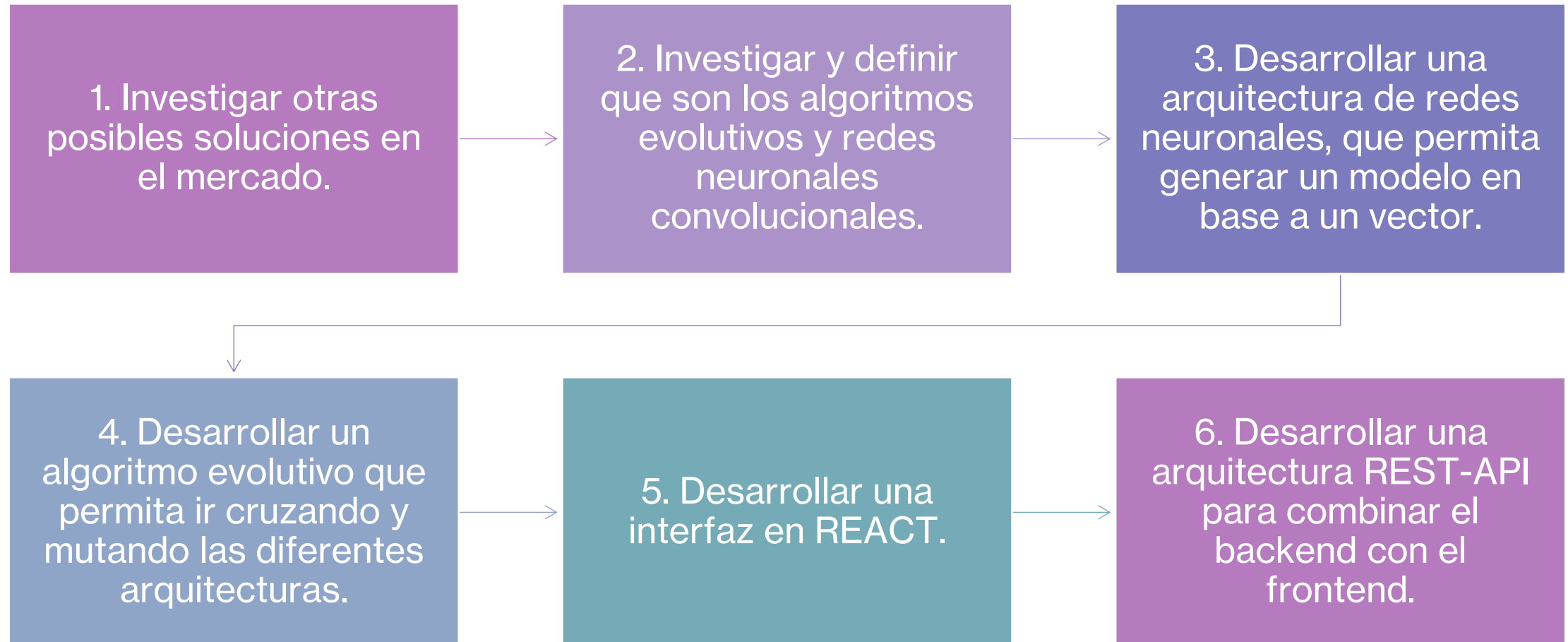
03  
Marco  
Teórico

04  
Marco  
Metodológico

05  
Análisis y  
Resultados

06  
Conclusiones

# Objetivos





# Marco Teórico

01

Introducción

02

Objetivos

Marco  
Teórico

03

04

Marco  
Metodológico

05

Análisis y  
Resultados

06

Conclusiones

# Algoritmos Genéticos

---



ESTE TIPO DE ALGORITMOS BUSCA  
RECREAR LA EVOLUCIÓN DE LAS  
ESPECIES DENTRO DE UN ALGORITMO  
SOFTWARE



DARWIN DESCRIBIÓ UN MECANISMO  
NATURAL PARA LA EVOLUCIÓN

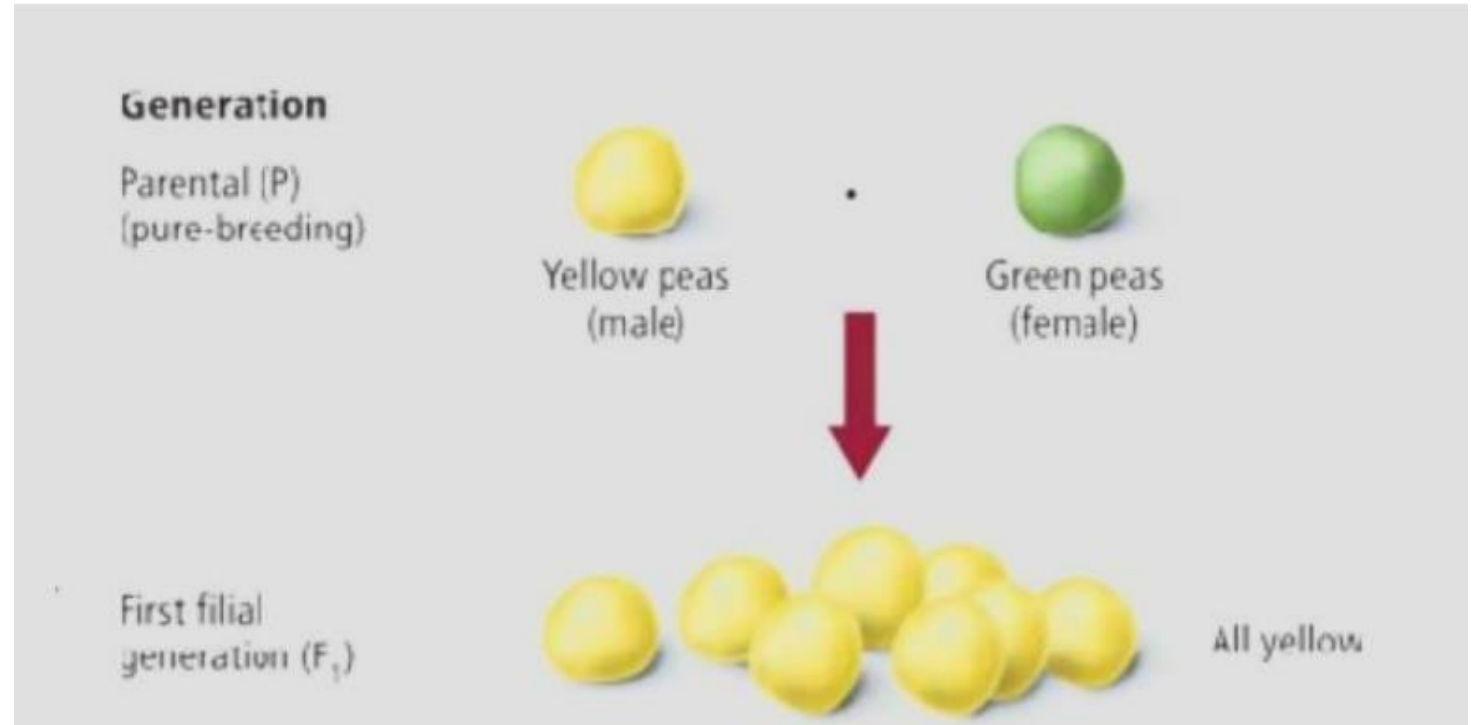


GREGOR MENDEL DESARROLLO LA  
TEORÍA DE LA GENÉTICA A FINALES DEL  
SIGLO XIX



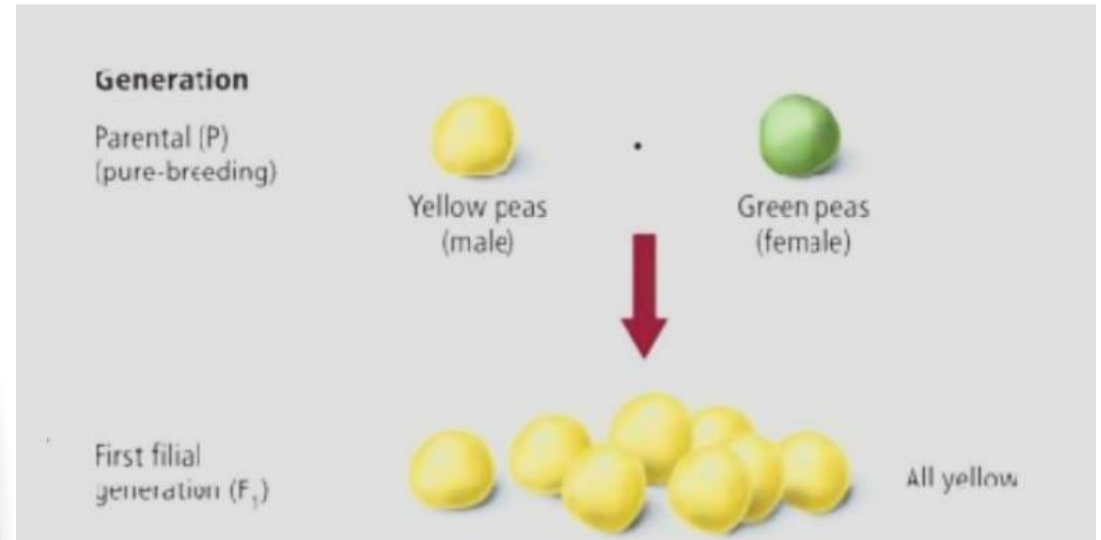
# Algoritmos Genéticos: Teoría de la genética

- Gregor Mendel, para desarrollar su teoría, experimento y cruzó diversos seres vivos como lo guisantes.
- Al cruzar los guisantes verdes puros y amarillos puros, la descendencia era amarilla






# Algoritmos Genéticos: Teoría de la genética

- Al cruzar esta segunda generación, que eran todos amarillos, aparecían verdes en una proporción de 3 a 1



## Algoritmos Genéticos: Genotipo

- Estos hechos, llevaron a Mendel a investigar y descubrir los fenotipos y genotipos.
- Para un gen, existe tanto su valor P, como su contrario, p.

Genotype		
<b><i>PP</i></b> (homozygous)	<b><i>Pp</i></b> (heterozygous)	<b><i>pp</i></b> (homozygous)
Phenotype		
 <b>Purple</b>	 <b>Purple</b>	 <b>White</b>

# Algoritmos Genéticos: Leyes de la genética



Ley de Uniformidad



Ley de Segregación



Ley de transmisión  
independiente

# Algoritmos Genéticos: Selección

---

- Selección proporcional
- Selección por torneo
- Selección por rango
- Selección elitista
- Selección estocástica

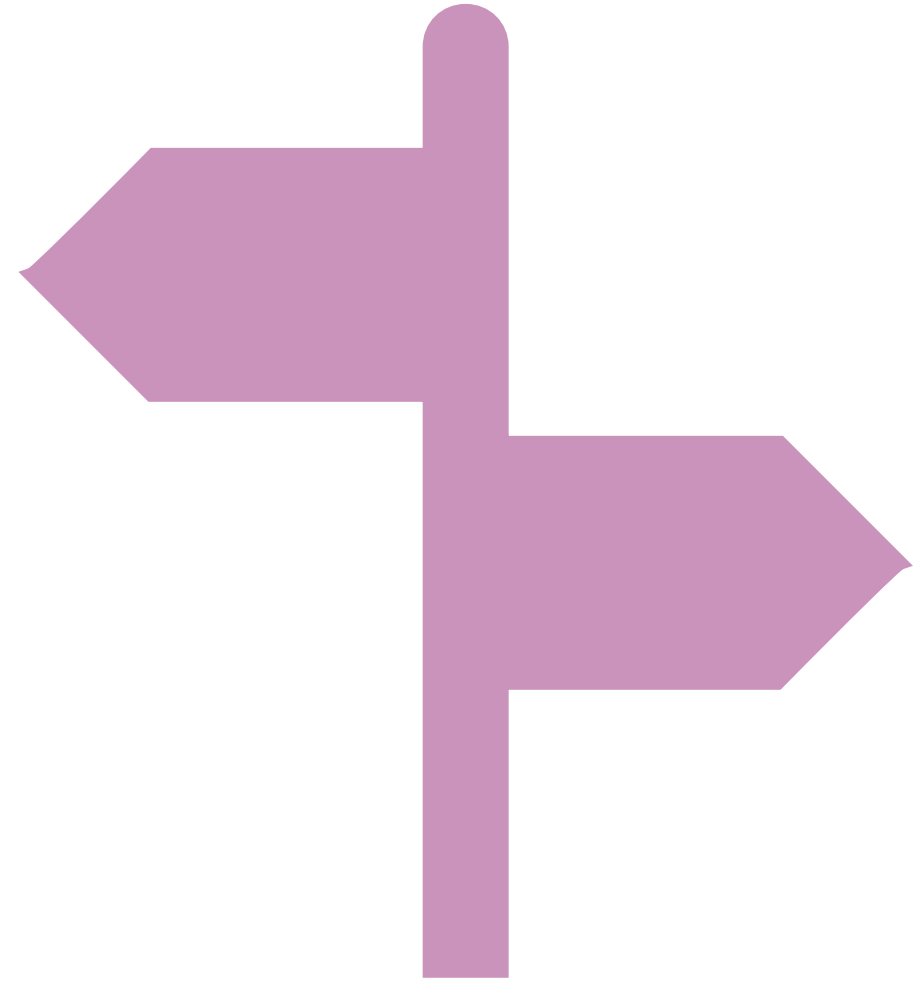




# Algoritmos Genéticos: Cruce

---

- Cruce de 1 punto
- Cruce de 2 puntos
- Cruce uniforme



# Algoritmos Genéticos: Mutación

---

- Tras haber generado un nuevo individuo en la fase de cruce, se “muta” para encontrar nuevas características que favorezcan su supervivencia



# Algoritmos Genéticos: Ejemplos y aplicaciones

---



Optimización de  
Redes y Sistemas



Finanzas



Gestión  
Automatizada



Inteligencia  
Artificial



Ingeniería y  
diseño



Medicina

# Deep Learning

---



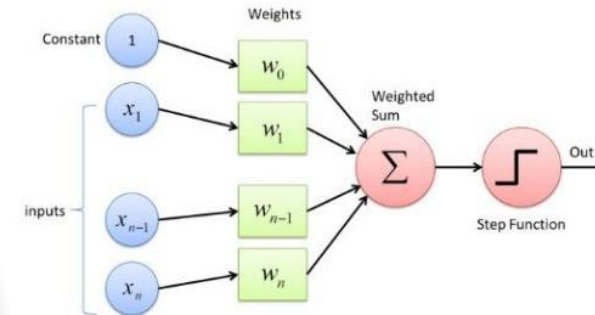
Subtipo de Machine Learning enfocado en el aprendizaje automático no supervisado



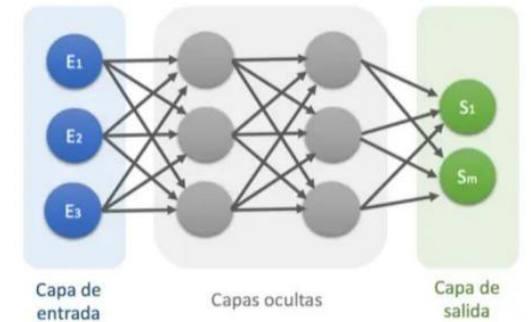
Intenta recrear el sistema neurológico humano

# ¿Qué es una red neuronal artificial?

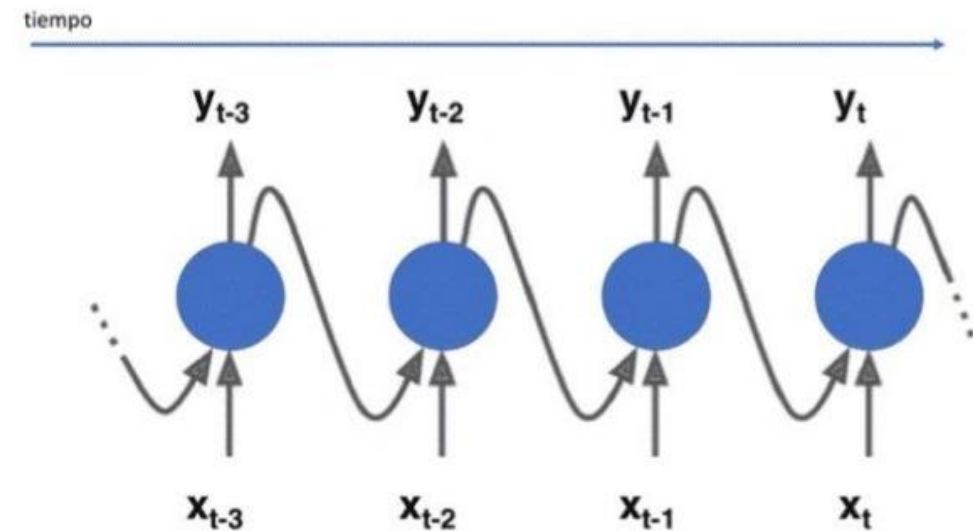
- Su nombre y estructura se basa en el funcionamiento del cerebro humano, tratando de imitar las neuronas biológicas
- Estas neuronas se conectan unas a otras formando capas, actualmente suele haber una capa de entrada, varias ocultas y una de salida



Percetrón simple



Percetrón multicapa



Red Recurrente



# ¿Cuándo surgieron las redes neuronales?

Creadas en las décadas de 1950 y 1960 por Frank Rosenblatt no ha parado de avanzar este campo hasta hoy en día.

En 1989 se desarrolló la Convolutional Neural Network (CNN)

En 2006 se crean las Deep Belief Networks

En 2017 presentan Las redes transformers

1950 - 1960

1986

1989

1997

2006

2014

2017

En 1986 se creó el algoritmo de Backpropagation

En 1997 se crearon las Long Short Term Memory

En 2014, se crea las Generative Adversarial Network

# REDES CNN

- Es la que se ha utilizado en este proyecto
- Dispone de varias capas ocultas especializadas en el reconocimiento de líneas curvas y hasta formas más complejas en las capas más profundas
- Necesitan una gran cantidad de imágenes para poder aprender, y a su vez de una gran cantidad de neuronas (solo para imágenes de 28x28 píxeles necesita 784 neuronas, 1352 si es a color).
- Entre las redes más conocidas destacan LeNet-5, GoogleNet, AlexNet, ZFNet, y VGG Net

# REDES CNN II

Su funcionamiento es el siguiente:

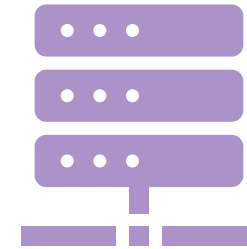
- Tras recibir la imagen, da un valor a cada pixel entre 0 y 255 y después lo divide entre 255.
- Crea grupos de pixeles cercanos o kernels, y calculará su producto escalar
- Después aplica la función de activación, actualmente la más usada es la función ReLU
- Se obtiene un mapa de características de la imagen original al que se aplica un subsampling o Max-Pooling.

# Funciones de coste

---



**Se encargan de determinar el error entre el valor estimado y el real para poder optimizar los parámetros y permitir a la red aprender**



**Destacan:**

**Cross Entropy Loss:** Valor entre 0 y 1, penalizando las predicciones erróneas y premiando las acertadas.

**Binary Cross Entropy:** Especializada en Clasificación Binaria

**MSE Loss:** utiliza el error cuadrático medio entre los elementos de la entrada y el objetivo

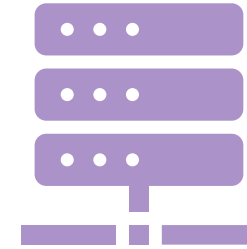
**Hinge Embedding Loss:** mide la pérdida dado un tensor  $x$  y otro  $y$

# Funciones de activación

---



Son el “como piensan” las neuronas, es decir; indican el resultado que da la neurona dada una entrada o conjunto de entradas.



**Destacan:**

**Función de Heaviside:** la primera utilizada por las redes neuronales, devuelve siempre 0 hasta superar un determinado umbral, entonces devolverá 1

**Función ReLU:** anula los valores negativos y no modifica los positivos

**Funciones Sigmoides:** mitiga los valores anormales, se caracteriza por estar delimitada por 2 asíntotas horizontales

**Función Logística:** convierte casi cualquier valor a uno entre 0 y 1

**Softmax:** especialización de la logístitca, que permite re-escalar las n dimensiones de una salida



# Redes Neuronales Automodelables

---

Proviene de la rama del Auto Machine Learning.

En 2010 comienza la investigación y el desarrollo en estas tecnologías

En 2016 presentan el uso del RL, con una arquitectura llamada controlador.

En 2018, se propone el uso de ENAS, que trata compartir parámetros de diferentes arquitecturas. Como controlador usan un LSTM

# Redes Neuronales Automodelables II

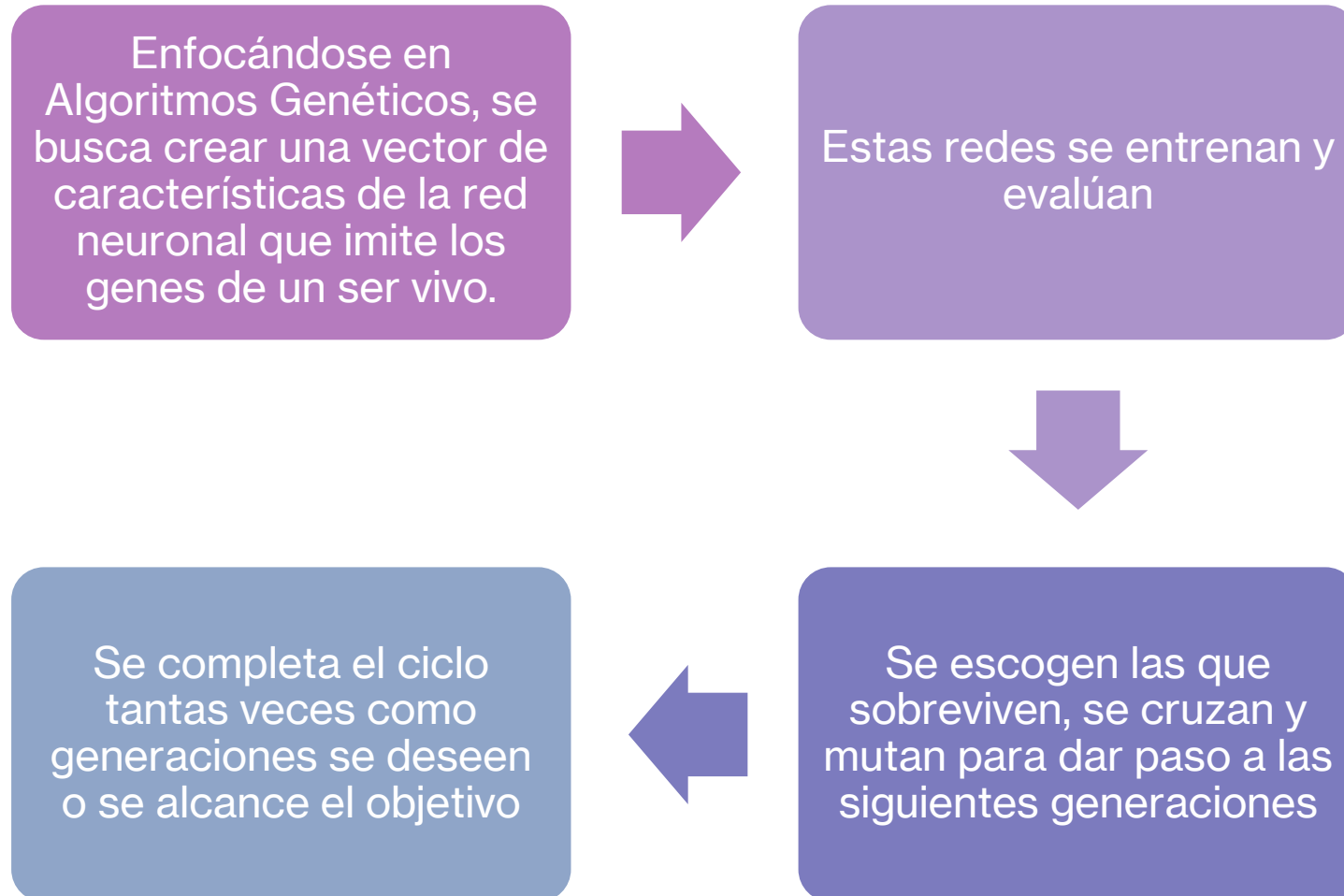
- Sin embargo, no solo se investiga el uso del RL, sino también en 2019 se presentó Autokeras, que utiliza:
  - NAS
  - Search Space
  - Optimización bayesiana
  - Procesamiento Automatizado de datos
  - Transfer Learning

# Redes Neuronales Automodelables III

- Además, se han probado otros enfoques como:
  - Algoritmos Genéticos
  - Aprendizaje por imitación
  - Modelos de reducción de dimensionalidad

# Redes Neuronales Automodelables IV

---



# Marco Metodológico



01

Introducción

02

Objetivos

03

Marco  
Teórico

04

Marco  
Metodológico

05

Análisis y  
Resultados

06

Conclusiones



# Técnicas Utilizadas

---

Backend: Python con  
Pytorch, Flask, y  
Json (guardar estado  
del algoritmo)

Frontend: ReactJS

Control de  
Versiones: Github

Dataset: CIFAR10

Arquitectura de  
microservicios y API  
- REST

Hardware: NVIDIA  
RTX3060 16GB  
VRAM, AMD 5  
5600X, 16GB RAM

# Endpoints

- Petición de búsqueda de arquitectura a un dataset
- Estado/Resultado de la búsqueda

# AutoModelizer

Introduce a dataset to discover its best architecture

## Search the best architecture

Introduce your data

Name  
carlos

Email  
bilbao@gmail.com

Number of descendency  
10

Number of epochs  
20

Number of classes  
10

☒ Is it splitted into train and test?

Seleccionar archivo

Work your magic →

## Check the results, please notice it could take several hours

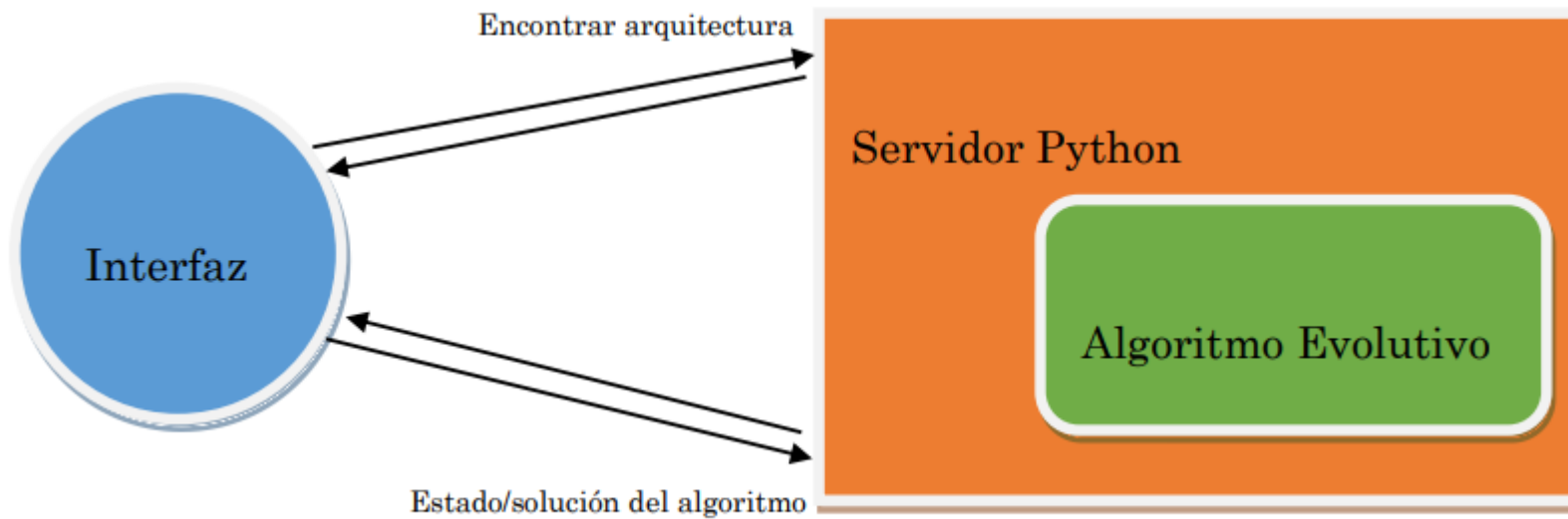
Name  
carlos

Email  
bilbao@gmail.com

check results →

**Learning Rate:** 0.00001  
**Number of Convolutional Layers:** 2  
**Kernel Sizes:** 1, 5  
**Filters:** 32, 128  
**Fully connected Layers:** 0  
**Dropout:** 0  
**Accuracy:** 0.5637

# Arquitectura software



# Población y Muestra

CIFAR 10

60.000  
imágenes a  
color

10 clases

6.000  
imágenes por  
clase

50.000 para  
entrenamiento

10.000 para  
evaluación

# Diseño Experimental

---



Desarrollo aislado en un  
jupyter notebook



Implementación del servidor

# Modelo de Análisis de datos

- Se mide la precisión del mejor modelo de cada generación para comprobar cómo evoluciona el algoritmo.
- Uso de matplotlib para crear gráficas que permitan una mejor interpretabilidad

!! Por limitaciones de hardware, se recogen los datos de 13 generaciones

```
Epoch 1/7: 100%|██████████| 391/391 [00:16<00:00, 24.34batch/s, training_loss=4.219]
Epoch 2/7: 100%|██████████| 391/391 [00:16<00:00, 24.42batch/s, training_loss=3.993]
Epoch 3/7: 100%|██████████| 391/391 [00:15<00:00, 24.70batch/s, training_loss=3.790]
Epoch 4/7: 100%|██████████| 391/391 [00:16<00:00, 24.42batch/s, training_loss=3.611]
Epoch 5/7: 100%|██████████| 391/391 [00:15<00:00, 24.69batch/s, training_loss=3.793]
Epoch 6/7: 100%|██████████| 391/391 [00:16<00:00, 24.39batch/s, training_loss=3.710]
Epoch 7/7: 100%|██████████| 391/391 [00:15<00:00, 24.81batch/s, training_loss=3.574]
{'filters': [64, 64], 'kernel_sizes': [7, 3], 'learning_rate': 1e-05, 'fully_connect': True}
Epoch 1/7: 100%|██████████| 391/391 [13:58<00:00, 2.14s/batch, training_loss=3.419]
Epoch 2/7: 100%|██████████| 391/391 [13:48<00:00, 2.12s/batch, training_loss=2.909]
Epoch 3/7: 100%|██████████| 391/391 [14:22<00:00, 2.21s/batch, training_loss=2.976]
Epoch 4/7: 100%|██████████| 391/391 [13:51<00:00, 2.13s/batch, training_loss=2.629]
Epoch 5/7: 100%|██████████| 391/391 [13:57<00:00, 2.14s/batch, training_loss=2.438]
Epoch 6/7: 100%|██████████| 391/391 [15:22<00:00, 2.36s/batch, training_loss=2.314]
Epoch 7/7: 100%|██████████| 391/391 [15:21<00:00, 2.36s/batch, training_loss=1.792]
```



# Análisis y Resultados

---

01

Introducción

02

Objetivos

03

Marco  
Teórico

04

Marco  
Metodológico

05

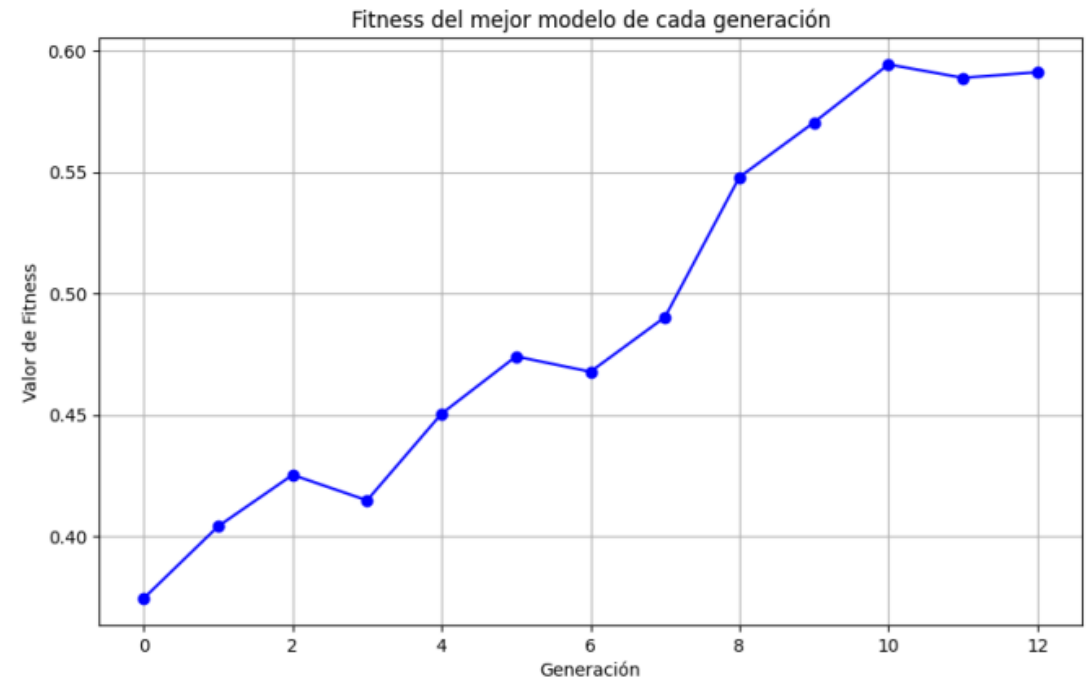
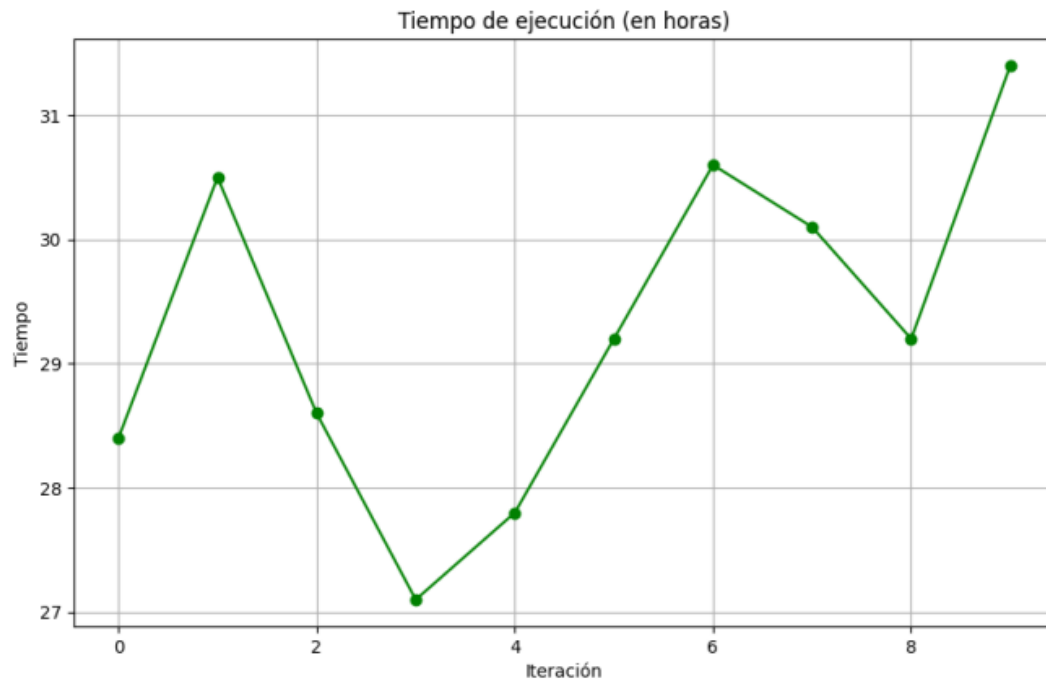
Análisis y  
Resultados

06

Conclusiones

# Resultados Obtenidos

- Tiempo de ejecución
- Evolución del fitness (media) a lo largo de la ejecución







# Análisis de los resultados

- El algoritmo, con este hardware, es lento, pero evoluciona adecuadamente hacia el objetivo
- Aun así, suponiendo que cada epoch es de 1 minuto, el tiempo total es del algoritmo es de 5000 minutos ( $1 \times 25$  epochs  $\times 10$  individuos  $\times 20$  generaciones), que son 83.33h o 3.47 días
- Un humano tardaría aproximadamente 42h, pero al trabajar 8h al día y no las 24 como el algoritmo, necesitaría 5.25 días



# Conclusiones

---

01

Introducción

02

Objetivos

03

Marco  
Teórico

04

Marco  
Metodológico

05

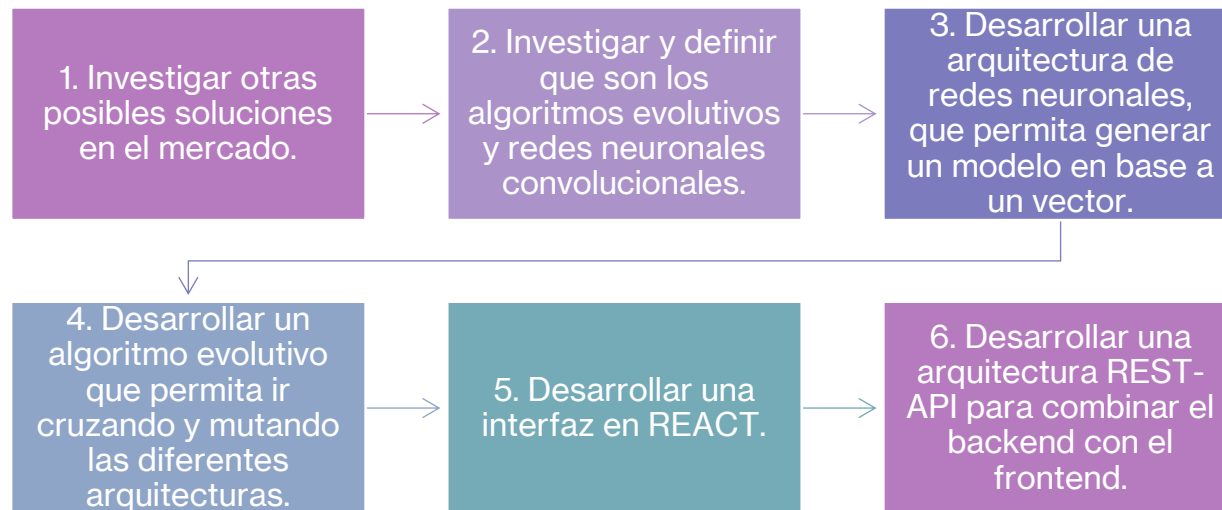
Análisis y  
Resultados

06

Conclusiones

# Objetivos

- Se ha cumplido con todos los objetivos del proyecto



```
127.0.0.1 - - [17/Jun/2024 21:38:15] "POST /result HTTP/1.1" 202 -
Datos eliminados si existían.
Parámetro 'bilbao@gmail.com' no encontrado.
127.0.0.1 - - [17/Jun/2024 21:38:58] "POST /result HTTP/1.1" 200 -
CUDA (GPU) está disponible en tu sistema.
127.0.0.1 - - [17/Jun/2024 21:40:25] "POST /upload HTTP/1.1" 202 -
*****
Generation: 0
*****
{'num_conv_layers': 2, 'filters': [64, 64], 'kernel_sizes': [5, 7], 'learning_rate': 0.1, 'fully_connected': 1, 'dropout': 0}
Epoch 1/20: 100% | 391/391 [00:39<00:00, 9.83batch/s, training_loss=58.391]
Epoch 2/20: 54% | 212/391 [00:18<00:16, 10.69batch/s, training_loss=32.074]
Parámetro 'bilbao@gmail.com' no encontrado.
127.0.0.1 - - [17/Jun/2024 21:41:24] "POST /result HTTP/1.1" 200 -
[epoch 2/20: 100% | 391/391 [00:34<00:00, 11.23batch/s, training_loss=27.130]
[epoch 3/20: 91% | 354/391 [00:29<00:07, 11.52batch/s, training_loss=41.051]
```

## AutoModelizer

Introduce a dataset to discover its best architecture

### Search the best architecture

Introduce your data

Name  
carlos

Email  
bilbao@gmail.com

check results -->

Number of descendency  
10

Number of epochs  
20

Number of classes  
10

☒ Is it splitted into train and test?

Seleccionar archivo

Work your magic -->

### Check the results, please notice it could take several hours

Name  
carlos

Email  
bilbao@gmail.com

check results -->

Learning Rate: 0.00001

Number of Convolutional Layers: 2

Kernel Sizes: 1, 5

Filters: 32, 128

Fully connected Layers: 0

Dropout: 0

Accuracy: 0.5637

39



---

# Futuras lineas de trabajo

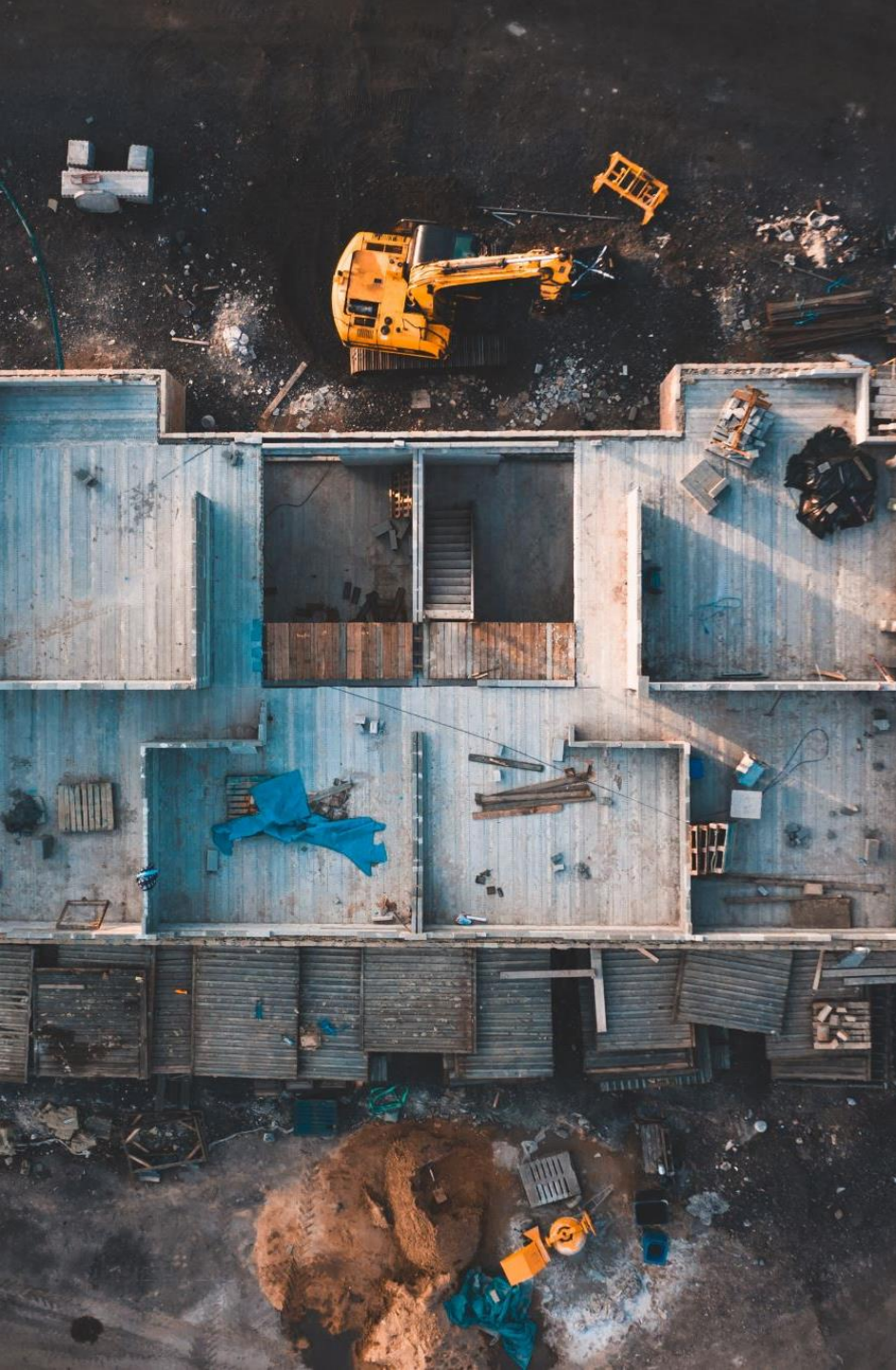
---

- Implementar un sistema distribuido para el entrenamiento de los individuos
- Adaptarlo a otro tipo de redes
- Seguimiento del entrenamiento
- Parar el algoritmo desde un endpoint

# Agradecimientos

- Me gustaría expresar mi agradecimiento a toda la universidad, en especial a mi tutor del TFM, Alberto Partida por su ayuda y constante revisión del estado del proyecto





---

# Enlace al repositorio

---

- Todo el proyecto, así como su instalación y uso está en Github

[Carlosbil/AutoModelizer](#)



# FIN

---

