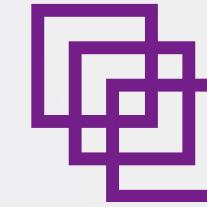




Universidad Politécnica del Estado de Morelos



# CLASIFICADOR DE TIPOS DE BASURA

Materia: Sistemas inteligentes

profesor:

Juan Paulo Sánchez Hernández

Integrantes:

García Solís Alejandro  
Reyes Ocampo Alexis  
Trujillo Vences Sergio Axel

8-D ITI

22 de Marzo de 2025

# Índice de Contenido

**01** Introducción

**02** Diseño de la propuesta

**03** Implementación

**04** Demostración

**05** Conclusiones

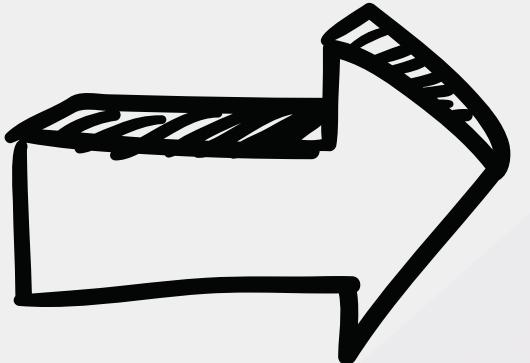
**06** Bibliografía

# INTRODUCCIÓN

Mala clasificación de  
residuos



Solución tecnológica

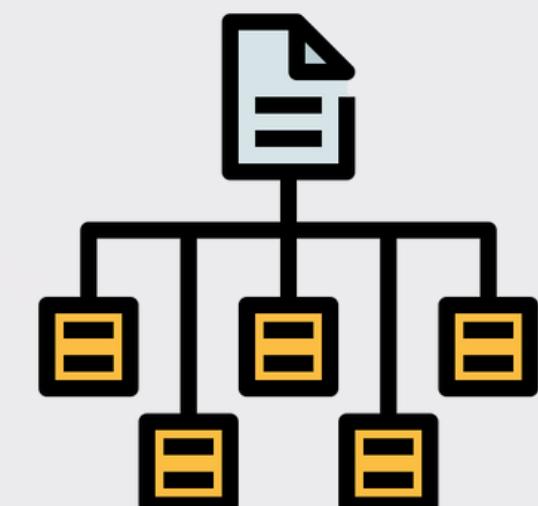
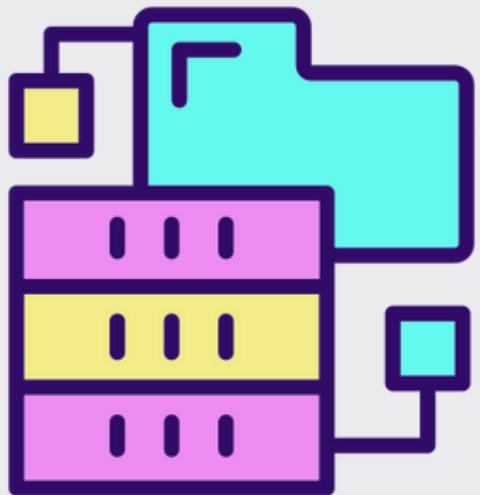


Clasificación automática



# DISEÑO DE LA PROPUESTA

- Interfaz: Se optó por desarrollar el proyecto en Google Colab debido a su facilidad de uso, integración con librerías de aprendizaje profundo y soporte para GPU.
- Datos de entrada: Imágenes de residuos reciclables, organizadas por categorías, obtenidas del dataset "Garbage Classification" disponible en Kaggle.
- Características del dataset: Contiene imágenes clasificadas en diversas categorías (papel, plástico, vidrio, etc.), con distintos formatos y resoluciones, que fueron redimensionadas a 150x150 px y normalizadas para su procesamiento.



# IMPLEMENTACIÓN

El modelo se desarrolló con TensorFlow/Keras, implementando capas convolucionales y de max-pooling para extraer características visuales.

## Primerº

Se utilizó ImageDataGenerator para cargar las imágenes, normalizar sus valores y dividirlas automáticamente en conjuntos de entrenamiento y validación, optimizando el preprocesamiento de datos para el modelo.

## Segundo Objetivo

Durante el desarrollo surgieron incompatibilidades con versiones de Python en equipos locales, resolviéndose al implementar el proyecto en Google Colab.

## Tercer Objetivo

El sistema logró una precisión competitiva en menos de 10 épocas, mostrando eficiencia en esta versión inicial del clasificador.

# DEMOSTRACIÓN



# CÓDIGO

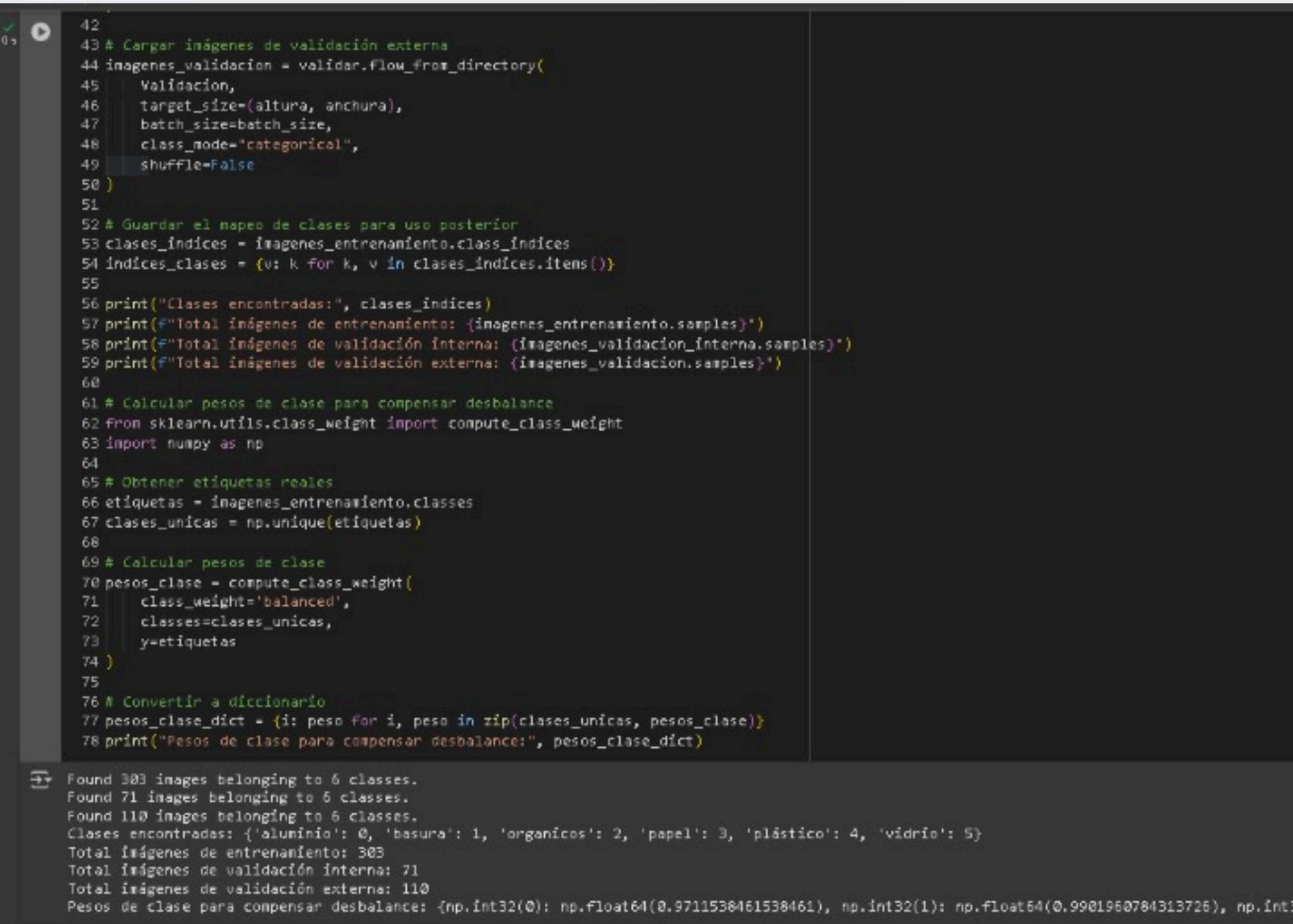
```
[29] 1 # Importar las librerías requeridas
2 import tensorflow as tf
3 from tensorflow.keras.preprocessing.image import ImageDataGenerator
4 from tensorflow.keras.applications import MobileNetV2
5 from tensorflow.keras.models import Sequential, Model, load_model
6 from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D, Input
7 from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
8 import numpy as np
9 import os
10
11 # Configurar memoria de GPU si está disponible
12 gpus = tf.config.experimental.list_physical_devices('GPU')
13 if gpus:
14     try:
15         for gpu in gpus:
16             tf.config.experimental.set_memory_growth(gpu, True)
17     except RuntimeError as e:
18         print(e)

[30] 1 # Definir la ruta de las imágenes
2 Entrenamiento = "/content/drive/MyDrive/MyEI/Imagenes/Entrenar"
3 Validacion = "/content/drive/MyDrive/MyEI/Imagenes/Validar"
4
5 # Definir los Hiperparámetros
6 epochas = 50
7 altura, anchura = 224, 224 # Tamaño estándar para MobileNetV2
8 batch_size = 16 # Reducido para mejor generalización
9 clases = 6 # orgánico, vidrio, metal, papel, plástico, basura
10
11 # Crear directorio para guardar modelos si no existe
12 modelo_dir = "/content/drive/MyDrive/MyEI/Imagenes/Modelo"
13 if not os.path.exists(modelo_dir):
14     os.makedirs(modelo_dir)
```

# CÓDIGO

```
[31] 1 # Preprocesamiento para MobileNetV2
2 from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
3
4 # Generador de datos con aumento para entrenamiento
5 entrenar = ImageDataGenerator(
6    preprocessing_function=preprocess_input,
7    rotation_range=40,
8    width_shift_range=0.2,
9    height_shift_range=0.2,
10   shear_range=0.2,
11   zoom_range=0.2,
12   horizontal_flip=True,
13   vertical_flip=True,
14   fill_mode='nearest',
15   validation_split=0.2 # Usar 20% de los datos de entrenamiento para validación
16 )
17
18 # Generador para validación (solo preprocesamiento)
19 validar = ImageDataGenerator(
20    preprocessing_function=preprocess_input
21 )
22
23 # Cargar imágenes de entrenamiento
24 imagenes_entrenamiento = entrenar.flow_from_directory(
25     Entrenamiento,
26     target_size=(altura, anchura),
27     batch_size=batch_size,
28     class_mode="categorical",
29     shuffle=True,
30     subset='training'
31 )
32
33 # Cargar imágenes de validación interna
34 imagenes_validacion_interna = entrenar.flow_from_directory(
35     Entrenamiento,
36     target_size=(altura, anchura),
37     batch_size=batch_size,
38     class_mode="categorical",
39     shuffle=True,
40     subset='validation'
41 )
42
```

# CÓDIGO



```
42
43 # Cargar imágenes de validación externa
44 imagenes_validacion = validar.flow_from_directory(
45     Validacion,
46     target_size=(altura, anchura),
47     batch_size=batch_size,
48     class_mode='categorical',
49     shuffle=False
50 )
51
52 # Guardar el mapeo de clases para uso posterior
53 clases_indices = imagenes_entrenamiento.class_indices
54 indices_clases = {v: k for k, v in clases_indices.items()}
55
56 print("Clases encontradas:", clases_indices)
57 print(f"Total imágenes de entrenamiento: {imagenes_entrenamiento.samples}")
58 print(f"Total imágenes de validación interna: {imagenes_validacion_interna.samples}")
59 print(f"Total imágenes de validación externa: {imagenes_validacion.samples}")
60
61 # Calcular pesos de clase para compensar desbalance
62 from sklearn.utils.class_weight import compute_class_weight
63 import numpy as np
64
65 # Obtener etiquetas reales
66 etiquetas = imagenes_entrenamiento.classes
67 clases_unicas = np.unique(etiquetas)
68
69 # Calcular pesos de clase
70 pesos_clase = compute_class_weight(
71     class_weight='balanced',
72     classes=clases_unicas,
73     y=etiquetas
74 )
75
76 # Convertir a diccionario
77 pesos_clase_dict = {i: peso for i, peso in zip(clases_unicas, pesos_clase)}
78 print("Pesos de clase para compensar desbalance:", pesos_clase_dict)

Found 303 images belonging to 6 classes.
Found 71 images belonging to 6 classes.
Found 110 images belonging to 6 classes.
Clases encontradas: {'aluminio': 0, 'basura': 1, 'organicos': 2, 'papel': 3, 'plástico': 4, 'vidrio': 5}
Total imágenes de entrenamiento: 303
Total imágenes de validación interna: 71
Total imágenes de validación externa: 110
Pesos de clase para compensar desbalance: {np.int32(0): np.float64(0.9711536461538461), np.int32(1): np.float64(0.9901960784313726), np.int32(2): np.float64(0.9901960784313726), np.int32(3): np.float64(0.9901960784313726), np.int32(4): np.float64(0.9901960784313726), np.int32(5): np.float64(0.9901960784313726)}
```

```

 1 from tensorflow.keras.utils import load_img, img_to_array
 2 from tensorflow.keras.models import load_model
 3 import numpy as np
 4 import json
 5 import os
 6
 7 def evaluar_imagen(ruta_imagen, modelo_path=None, modelo=None):
 8     """
 9         Evalúa una imagen y muestra solo el resultado de la clasificación
10     """
11     # Cargar el modelo si no se proporciona
12     if modelo is None and modelo_path:
13         modelo = load_model(modelo_path)
14
15     # Cargar mapeo de clases
16     clases_path = os.path.join(os.path.dirname(modelo_path), "clases_indices.json")
17     try:
18         with open(clases_path, "r") as f:
19             clases_indices = json.load(f)
20             indices_clases = {int(v): k for k, v in clases_indices.items()}
21     except:
22         # Mapeo manual si no se encuentra el archivo
23         indices_clases = {
24             0: 'organico',
25             1: 'vidrio',
26             2: 'metal',
27             3: 'papel',
28             4: 'plastico',
29             5: 'basura'
30         }
31
32     # Preprocesar la imagen
33     imagen = load_img(ruta_imagen, target_size=(224, 224))
34     img_array = img_to_array(imagen)
35     img_array = np.expand_dims(img_array, axis=0)
36     img_array = preprocess_input(img_array) # Preprocesamiento específico para MobileNetV2
37
38     # Predicir
39     prediccion = modelo.predict(img_array, verbose=0)
40
41     # Obtener la clase con mayor probabilidad
42     clase_idx = np.argmax(prediccion[0])
43     clase_nombre = indices_clases[clase_idx]
44     probabilidad = prediccion[0][clase_idx]
45
46     # Mostrar solo el resultado
47     print(f"Tipo de residuo: {clase_nombre.upper()} (confianza: {probabilidad:.2%})")
48
49     return clase_nombre, probabilidad
50
51 # Ejemplo de uso
52 modelo_path = os.path.join(modelo_dir, "modelo_final_transfer.h5")

```

```

 1 def evaluar_directorio(directorio, modelo_path, num_imagenes=None):
 2     """
 3         Evalúa todas las imágenes en un directorio y muestra estadísticas simplificadas
 4     """
 5     # Cargar el modelo una sola vez
 6     modelo = load_model(modelo_path)
 7
 8     # Obtener lista de imágenes
 9     imagenes = []
10     for root, _, files in os.walk(directorio):
11         for file in files:
12             if file.lower().endswith(('.png', '.jpg', '.jpeg')):
13                 imagenes.append(os.path.join(root, file))
14
15     # Limitar número de imágenes si se especifica
16     if num_imagenes and len(imagenes) > num_imagenes:
17         imagenes = imagenes[:num_imagenes]
18
19     # Estadísticas
20     total = len(imagenes)
21     resultados = {}
22
23     # Procesar cada imagen
24     for i, img_path in enumerate(imagenes):
25         print(f"Imagen {i+1}/{total}: {os.path.basename(img_path)}")
26
27         # Obtener clase real del nombre del directorio
28         clase_real = os.path.basename(os.path.dirname(img_path)).lower()
29
30         # Evaluar imagen
31         clase_pred, _ = evaluar_imagen(img_path, modelo_path=modelo_path, modelo=modelo)
32
33         # Registrar resultado
34         if clase_real not in resultados:
35             resultados[clase_real] = {'total': 0, 'correctas': 0}
36
37             resultados[clase_real]['total'] += 1
38             if clase_real == clase_pred.lower():
39                 resultados[clase_real]['correctas'] += 1
40
41     # Mostrar resultados simplificados
42     print("\n--- RESULTADOS DE EVALUACIÓN ---")
43     print(f"Total de imágenes evaluadas: {total}")
44
45     total_correctas = sum(r['correctas'] for r in resultados.values())
46     precision_global = total_correctas / total if total > 0 else 0
47
48     print(f"Precision global: {precision_global:.2%}")
49     print("\nResultados por clase:")
50
51     for clase, stats in resultados.items():
52         precision = stats['correctas'] / stats['total'] if stats['total'] > 0 else 0
53         print(f" {clase}: {precision:.2%} ({stats['correctas']}/{stats['total']})")
54
55 # Ejemplo de uso
56 directorio_validacion = "/content/drive/MyDrive/MyEI/Imagenes/Validar"
57 modelo_path = os.path.join(modelo_dir, "modelo_final_transfer.h5")

```

```
✓ 1 from google.colab import files  
 2 import io  
 3  
 4 def interfac_clasificacion_simple():  
 5     """Crea una interfaz simple para clasificar imágenes en Google Colab"""\n 6  
 7     # Cargar el modelo  
 8     modelo_path = os.path.join(modelo_dir, "modelo_final_transfer.h5")  
 9     modelo = load_model(modelo_path)  
10  
11    print("==> CLASIFICADOR DE RESIDUOS ==>")  
12    print("Selecciona una imagen para clasificar...")  
13  
14    # Subir imagen  
15    uploaded = files.upload()  
16  
17    # Procesar cada imagen subida  
18    for filename, content in uploaded.items():  
19        # Guardar temporalmente  
20        temp_path = f"/tmp/{filename}"  
21        with open(temp_path, 'wb') as f:  
22            f.write(content)  
23  
24        # Clasificar y mostrar solo el resultado  
25        print(f"\nResultado para {filename}:")  
26        evaluar_imagen(temp_path, modelo_path=modelo_path, modelo=modelo)  
27  
28 # Ejecutar la interfaz  
29 interfac_clasificacion_simple()
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.  
==> CLASIFICADOR DE RESIDUOS ==>  
Selecciona una imagen para clasificar...  
Elegir archivo: 62.jpg  
* 62.jpg(image/jpeg)-19117 bytes, last modified:1/1/1980-100% done  
Saving 62.jpg to 62.jpg  
  
Resultado para 62.jpg:  
Tipo de residuo: ALUMINIO (confianza: 100.00%)
```

# CONCLUSIONES

Logros



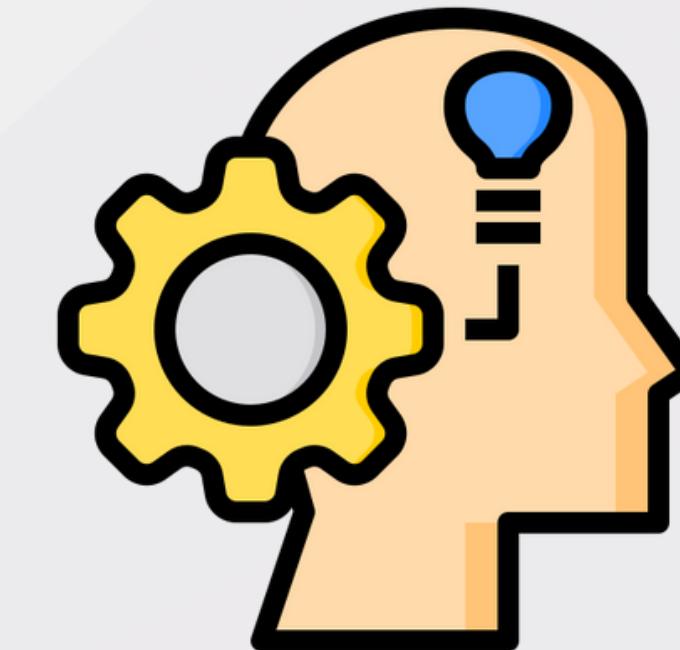
Sistema funcional

Dificultades



Complejidad

Aprendizajes



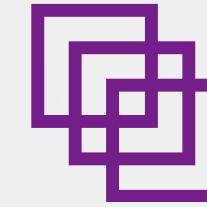
Nuevas tecnologías

# BIBLIOGRAFÍA

- Adam | Interactive Chaos. (s. f.). <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/adam>
- Daniel. (2023, 30 octubre). *Convolutional Neural Network: definición y funcionamiento.* Formación En Ciencia de Datos | DataScientest.com. <https://datascientest.com/es/convolutional-neural-network-es>
- TensorFlow. (s. f.). *TensorFlow.* <https://www.tensorflow.org/?hl=es-419>
- Ultralytics. (s. f.). *Adam Optimizer - Learn how the Adam optimizer powers efficient neural network training with adaptive learning rates, momentum, and real-world applications in AI.* <https://www.ultralytics.com/es/glossary/adam-optimizer>



Universidad Politécnica del Estado de Morelos



# MUCHAS GRACIAS