

**U**. **Internacional Aguascalientes**

**Maestría en Inteligencia Artificial para la Transformación Digital**

**RAZONAMIENTO INTELIGENTE**

**Andrés Hernández Lara**

**Dra. Claudia Andrea Vidales Basurto**

**2.1. Clasificación para la detección.**

**Aguascalientes, Ags., a 3 de octubre de 2025**

Tabla de contenido

[INTRODUCCIÓN 3](#_Toc210341479)

[INSTRUCCIONES. 3](#_Toc210341480)

[ENLACE A REPOSITORIO O CARPETA COMPARTIDA. 3](#_Toc210341481)

[INSTRUCCIONES DE COMPILACIÓN O EJECUCIÓN 3](#_Toc210341482)

[NOMBRE Y VERSIÓN DE BIBLIOTECAS UTILIZADAS. 3](#_Toc210341483)

[INSTALACIÓN DE LIBRERIAS 3](#_Toc210341484)

[SISTEMA OPERATIVO UTILIZADO 4](#_Toc210341485)

[Desarrollo de una red neuronal CNN 4](#_Toc210341486)

[PLANTEAMIENTO 4](#_Toc210341487)

[PARA NUESTRO PROYECTO 5](#_Toc210341488)

[ARQUITECTURA 5](#_Toc210341489)

[CODIGO 5](#_Toc210341490)

[PREPARACION DEL DATASET 10](#_Toc210341491)

[SALIDAS 12](#_Toc210341492)

[CONCLUSIÓN 23](#_Toc210341493)

[REFERENCIAS (FORMATO APA) 24](#_Toc210341494)

## INTRODUCCIÓN

Las redes neuronales convolucionales (CNNs) son un tipo especial de redes neuronales diseñadas para procesar información que tiene una estructura espacial, como las imágenes.

A diferencia de las redes tradicionales, que tratan cada entrada de forma independiente, las CNNs explotan la correlación local entre píxeles adyacentes. Esto les permite identificar patrones y características relevantes, como bordes, texturas o formas, que se combinan en niveles jerárquicos para reconocer objetos complejos. Cada capa convolucional aplica pequeños filtros a la imagen, generando mapas de activación que resaltan ciertos patrones. Posteriormente, técnicas como el pooling reducen la dimensionalidad, manteniendo solo la información más significativa y haciendo la red más robusta a cambios de posición o escala.

Las CNNs son especialmente útiles en visión por computadora porque pueden aprender directamente de los datos sin necesidad de diseñar manualmente características. Esto permite aplicaciones que van desde el reconocimiento de objetos en fotos hasta sistemas de conducción autónoma o análisis médico.

# INSTRUCCIONES.

Partiendo de las nociones teóricas y prácticas vistas en la unidad, el alumno va a diseñar y desarrollar una red aplicada en el área de su interés que sea capaz de clasificar para la detección de alguna característica o parámetro. Tanto el diseño como el código deben ser entregados describiendo detalladamente cada uno de los procesos involucrados, así como la justificación y pertinencia de su implementación.

## ENLACE A REPOSITORIO O CARPETA COMPARTIDA.

Repositorio con el material requerido para visualizar el ejemplo.

<https://github.com/andreshl/redc_CNN>

## INSTRUCCIONES DE COMPILACIÓN O EJECUCIÓN

El presente código es un **script python**, Se desarrollo corrigió y ejecuto en el ide **VSCode**, corriendo de forma local utilizando **Python 3.11.0**

## NOMBRE Y VERSIÓN DE BIBLIOTECAS UTILIZADAS.

Dentro del notebook se incluye el comando para la instalación de las librerías, anexo en una celda.

## INSTALACIÓN DE LIBRERIAS

**pip install torch torchvision numpy matplotlib scikit-learn**

## SISTEMA OPERATIVO UTILIZADO

El sistema operativo utilizado fue Windows 11, en instalación por parte de mi unidad de trabajo, se **censuran** los siguientes datos (**clave de versión, numero de compilación y dueño del licenciamiento**)



## Desarrollo de una red neuronal CNN

### PLANTEAMIENTO

En el contexto de videojuegos, uno de los objetivos es **detectar enemigos en tiempo real** ya sea a partir de capturas de pantalla o flujos de video, con la finalidad de realizar tareas mas complejas en el desarrollo de parches, pruebas automatizadas o mods, Cada enemigo presenta patrones visuales que lo distinguen del entorno: forma, color, texturas o posición relativa. Las CNNs son ideales para esto porque pueden:

**Aprender automáticamente estos patrones** a partir de ejemplos. No necesitas decirle explícitamente qué es un enemigo: la red aprende de imágenes etiquetadas.

**Generalizar a nuevos escenarios**: aunque el entorno del juego cambie, la red puede reconocer al enemigo gracias a las características visuales que aprendió.

**Procesar imágenes en tiempo real** si se optimiza la red y se ejecuta en una GPU.

Tu arquitectura busca precisamente esto: diferenciar entre imágenes que contienen enemigos y aquellas que no.

### PARA NUESTRO PROYECTO

Se busca que, a través de la detección de patrones, logremos detectar al enemigo, a partir de diversas capturas de pantalla de un video de una partida y evaluar la proximidad real de la detección, pudiendo con esto hacer otros procesos con los posibles resultados, a continuación, se explica la arquitectura de la red trabajada

### ARQUITECTURA

La clase EnemyCNN, es un ejemplo de CNN aplicada a clasificación binaria de imágenes:

1. **Tres capas convolucionales (conv1, conv2, conv3)**: cada una seguida de una activación ReLU y un pooling 2×2. Esto permite que la red vaya detectando patrones cada vez más complejos y abstractos en las imágenes, desde bordes hasta formas completas de enemigos.
2. **Flatten y capas fully connected (fc1, fc2)**: después de extraer las características espaciales, se aplanan los mapas de activación y se combinan mediante una capa densa de 512 neuronas. Esto permite que la red integre toda la información para realizar la decisión final.
3. **Dropout**: previene el sobreajuste, ayudando a que la red generalice mejor a nuevas imágenes.
4. **Sigmoid**: al final, produce un valor entre 0 y 1, indicando la probabilidad de que la imagen contenga un enemigo (clase “enemy”) o no (“n\_enemy”).

La red aprende **qué hace que un enemigo se vea diferente del fondo o de otros objetos**, y luego combina esa información para tomar una decisión binaria.

El uso de **imágenes “n\_enemy”** (las cuales no contienen el patron buscado como enemigo) es fundamental, porque ayuda a la red a comprender **qué NO es un enemigo**, evitando falsos positivos. aplicaremos CNNs para **extraer patrones visuales jerárquicos** de los enemigos en un juego, entrenando la red con ejemplos positivos y negativos, y luego evaluando su capacidad de detectar correctamente estos patrones en nuevas imágenes.

Esto es exactamente el flujo que usan sistemas de detección automática en juegos o análisis de video.

### CODIGO

|  |
| --- |
| # ===============================  # 0. Librerias requeridas  # =============================== |
| import torch  import torch.nn as nn  import torch.optim as optim  from torchvision import datasets, transforms  from torch.utils.data import DataLoader, WeightedRandomSampler  import numpy as np  import os |

|  |
| --- |
| # ===============================  # 1. Arquitectura CNN  # =============================== |
| class EnemyCNN(nn.Module):      def \_\_init\_\_(self):          super(EnemyCNN, self).\_\_init\_\_()          self.conv1 = nn.Conv2d(3, 32, 3, padding=1)          self.conv2 = nn.Conv2d(32, 64, 3, padding=1)          self.conv3 = nn.Conv2d(64, 128, 3, padding=1)          self.pool = nn.MaxPool2d(2,2)            self.fc1 = nn.Linear(128\*16\*16, 512)          self.fc2 = nn.Linear(512, 1)            self.relu = nn.ReLU()          self.dropout = nn.Dropout(0.5)          self.sigmoid = nn.Sigmoid()        def forward(self,x):          x = self.pool(self.relu(self.conv1(x)))          x = self.pool(self.relu(self.conv2(x)))          x = self.pool(self.relu(self.conv3(x)))          x = x.view(x.size(0), -1)          x = self.relu(self.fc1(x))          x = self.dropout(x)          x = self.fc2(x)          return self.sigmoid(x) |

|  |
| --- |
| # ===============================  # 2. Configuración dataset  # =============================== |
| dataset\_path = os.path.join(os.path.dirname(\_\_file\_\_), "dataset")  train\_transform = transforms.Compose([      transforms.Resize((128,128)),      transforms.RandomHorizontalFlip(),      transforms.RandomRotation(10),      transforms.ToTensor(),      transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))  ])  test\_transform = transforms.Compose([      transforms.Resize((128,128)),      transforms.ToTensor(),      transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))  ])  train\_dataset = datasets.ImageFolder(os.path.join(dataset\_path,"train"), transform=train\_transform)  test\_dataset = datasets.ImageFolder(os.path.join(dataset\_path,"test"), transform=test\_transform) |

|  |
| --- |
| # ===============================  # 2a. WeightedRandomSampler para balancear clases  # =============================== |
| targets = [label for \_, label in train\_dataset]  class\_counts = np.bincount(targets)  class\_weights = 1. / class\_counts  samples\_weights = [class\_weights[t] for t in targets]  sampler = WeightedRandomSampler(samples\_weights, num\_samples=len(samples\_weights), replacement=True)  train\_loader = DataLoader(train\_dataset, batch\_size=32, sampler=sampler)  test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False)  print("Train clases:", train\_dataset.classes)  print("Test clases:", test\_dataset.classes) |

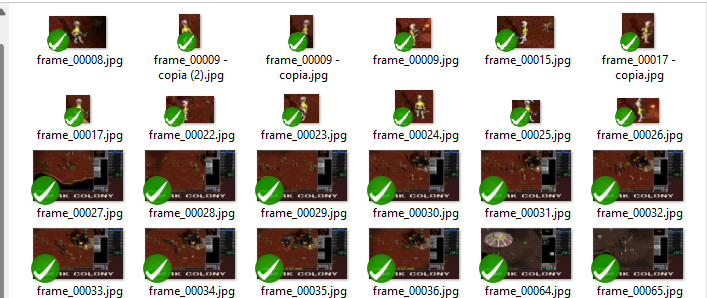
|  |
| --- |
| # ===============================  # 3. Entrenamiento  # =============================== |
| device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  model = EnemyCNN().to(device)  criterion = nn.BCELoss()  optimizer = optim.Adam(model.parameters(), lr=0.001)  epochs = 10  for epoch in range(epochs):      model.train()      running\_loss = 0.0      class\_correct = {cls: 0 for cls in train\_dataset.classes}      class\_total = {cls: 0 for cls in train\_dataset.classes}        for images, labels in train\_loader:          images = images.to(device)          labels = labels.to(device).float().unsqueeze(1)            optimizer.zero\_grad()          outputs = model(images)          loss = criterion(outputs, labels)          loss.backward()          optimizer.step()            running\_loss += loss.item()            # Conteo por clase          predicted = (outputs > 0.5).float()          for i in range(len(labels)):              true\_label = int(labels[i].item())              pred\_label = int(predicted[i].item())              class\_total[train\_dataset.classes[true\_label]] += 1              if true\_label == pred\_label:                  class\_correct[train\_dataset.classes[true\_label]] += 1        print(f"Epoch {epoch+1}/{epochs}, Loss: {running\_loss/len(train\_loader):.4f}")      for cls in train\_dataset.classes:          if class\_total[cls] > 0:              acc = 100\*class\_correct[cls]/class\_total[cls]              print(f"  📌 Train clase '{cls}': {acc:.2f}% ({class\_correct[cls]}/{class\_total[cls]})") |

|  |
| --- |
| # ===============================  # 4. Evaluación en test  # =============================== |
| idx\_to\_class = {v:k for k,v in train\_dataset.class\_to\_idx.items()}  model.eval()  correct, total = 0,0  class\_correct = {cls:0 for cls in test\_dataset.classes}  class\_total = {cls:0 for cls in test\_dataset.classes}  with torch.no\_grad():      for images, labels in test\_loader:          images = images.to(device)          labels = labels.to(device).float().unsqueeze(1)          outputs = model(images)          predicted = (outputs > 0.5).float()            total += labels.size(0)          correct += (predicted == labels).sum().item()            for i in range(len(labels)):              true\_label = int(labels[i].item())              pred\_label = int(predicted[i].item())              class\_total[idx\_to\_class[true\_label]] += 1              if true\_label == pred\_label:                  class\_correct[idx\_to\_class[true\_label]] += 1    print(f"\n🔎 Precisión total en test: {100\*correct/total:.2f}%")  for cls in test\_dataset.classes:      if class\_total[cls]>0:          acc = 100\*class\_correct[cls]/class\_total[cls]          print(f" Clase '{cls}': {acc:.2f}% ({class\_correct[cls]}/{class\_total[cls]})")  import matplotlib.pyplot as plt  import numpy as np  # Diccionario: índice de clase → nombre de clase (enemigos / no\_enemigos)  idx\_to\_class = {v: k for k, v in train\_dataset.class\_to\_idx.items()}  import matplotlib.pyplot as plt  model.eval()  with torch.no\_grad():      for i, (images, labels) in enumerate(test\_loader):          images = images.to(device)          labels = labels.to(device).float().unsqueeze(1)          outputs = model(images)          predicted = (outputs > 0.5).float()          for j in range(images.size(0)):              true\_label = int(labels[j].item())              pred\_label = int(predicted[j].item())              print(f"Imagen {i\*test\_loader.batch\_size + j}: "                    f"Real = {idx\_to\_class[true\_label]}, "                    f"Predicho = {idx\_to\_class[pred\_label]}")    from sklearn.metrics import classification\_report, confusion\_matrix  import numpy as np  # Diccionario de índice → nombre de clase  idx\_to\_class = {v: k for k, v in train\_dataset.class\_to\_idx.items()}  all\_preds = []  all\_labels = []  model.eval()  with torch.no\_grad():      for images, labels in test\_loader:          images = images.to(device)          labels = labels.to(device).float().unsqueeze(1)          outputs = model(images)          predicted = (outputs > 0.5).float()          all\_preds.extend(predicted.cpu().numpy())          all\_labels.extend(labels.cpu().numpy())  # Convertir a enteros  all\_preds = np.array(all\_preds).astype(int).flatten()  all\_labels = np.array(all\_labels).astype(int).flatten()  print("\n📊 Reporte de clasificación por clase:")  print(classification\_report(all\_labels, all\_preds, target\_names=list(idx\_to\_class.values())))  print("📌 Matriz de confusión:")  print(confusion\_matrix(all\_labels, all\_preds)) |

### PREPARACION DEL DATASET

Antes de comenzar, con el análisis al conjunto de salidas, explicaremos un poco como generamos el dataset para entrenar la red,

Para este proceso se tomaron las imágenes a partir de un video en el cual generamos muchos frames los cuales se van a utilizar para la detección de nuestros patrones en este caso solamente utilizaremos un solo patrón para garantizar y tener un poco más de control sobre las nociones de la red neuronal

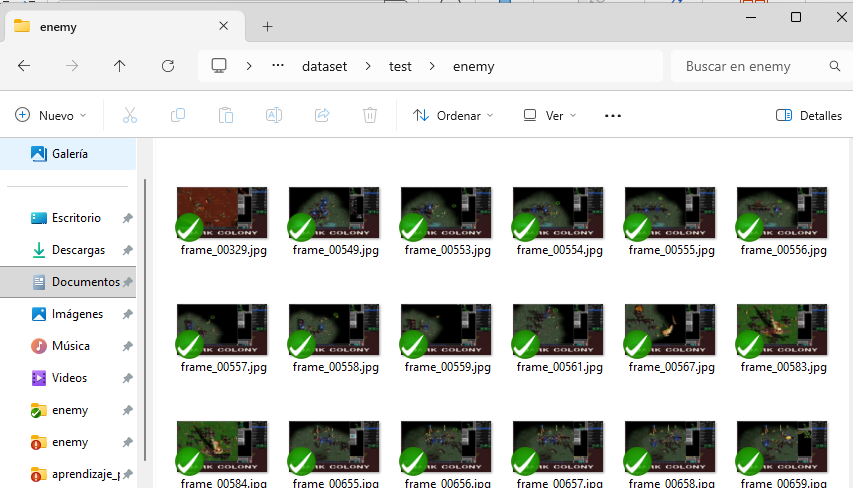


Como podemos notar algunos de los frames que se están recuperando se recortaron para lograr una mejor aproximación hacia el patrón buscado repito en este caso sólo se está buscando el patrón correspondiente al enemigo blanco con amarillo que corresponde a un alienígena.

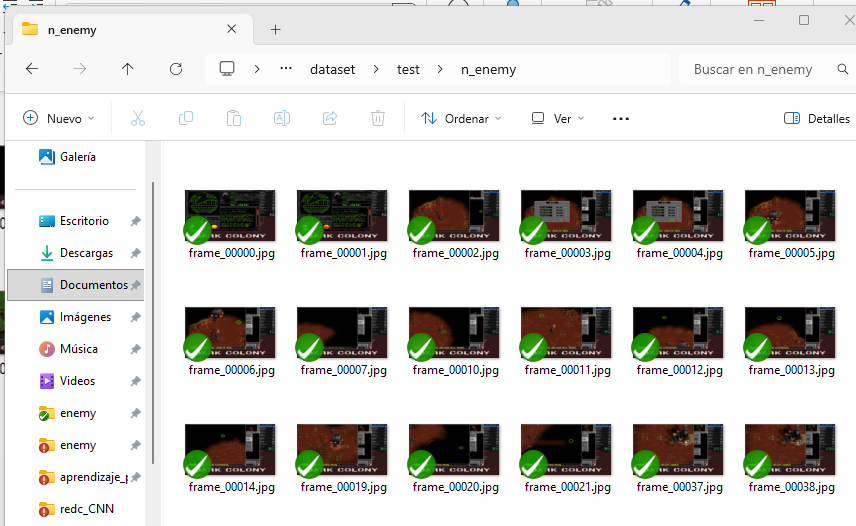
Dentro de nuestras imágenes vamos a encontrar muchísimos patrones, estos patrones corresponderán a soldados edificios opciones de menú capturas de Instrucciones en este caso ignoraremos todas ellas colocando algunas incluso otras variaciones de alienígenas dentro de nuestra carpeta n\_enemy enfocaremos solo en el patrón del alien blanco con amarillo.

Se realizaron diversas ejecuciones y pruebas dentro de las cuales se hicieron algunos cambios de parámetros a continuación colocó algunas imágenes de las imágenes de entrenamiento y las imágenes de prueba las cuales no se encuentran dentro del mismo conjunto.

Enemy de test



N\_enemy de test



### SALIDAS

A continuación, analizaremos diferentes salidas las cuales fueron en diferentes puntos de ejecución intercambiando parámetros como el número de generaciones o haciendo ajustes dentro de las imágenes de entrenamiento

|  |
| --- |
| **EJECUCION 1** |
| Nuestra primera ejecución se realizará sobre el código inicial con una cantidad de generaciones de 10 en esta se utilizó el mismo conjunto de imágenes final en la cual aparecen varios patrones con el enemigo y ha recordado junto con algunas en las cuales hay múltiples de enemigos, se espera que la red aprenda los patrones de ambas situaciones en las cuales el enemigo se encuentra combatiendo, descansando O en estado IDLE de acuerdo con la nomenclatura del sector. |
| Microsoft Windows [Versión 10.0.26100.6584]  (c) Microsoft Corporation. Todos los derechos reservados.  C:\Users\lilia\OneDrive\Escritorio\ahl\Mai\redc\_CNN>py CNN.py  Train clases: ['enemy', 'n\_enemy']  Test clases: ['enemy', 'n\_enemy']  Epoch 1/10, Loss: 0.8626  📌 Train clase 'enemy': 40.59% (41/101)  📌 Train clase 'n\_enemy': 60.98% (50/82)  Epoch 2/10, Loss: 0.6870  📌 Train clase 'enemy': 98.99% (98/99)  📌 Train clase 'n\_enemy': 0.00% (0/84)  Epoch 3/10, Loss: 0.7002  📌 Train clase 'enemy': 51.65% (47/91)  📌 Train clase 'n\_enemy': 58.70% (54/92)  Epoch 4/10, Loss: 0.6434  📌 Train clase 'enemy': 81.63% (80/98)  📌 Train clase 'n\_enemy': 29.41% (25/85)  Epoch 5/10, Loss: 0.6087  📌 Train clase 'enemy': 28.05% (23/82)  📌 Train clase 'n\_enemy': 96.04% (97/101)  Epoch 6/10, Loss: 0.5293  📌 Train clase 'enemy': 45.05% (41/91)  📌 Train clase 'n\_enemy': 98.91% (91/92)  Epoch 7/10, Loss: 0.5214  📌 Train clase 'enemy': 57.28% (59/103)  📌 Train clase 'n\_enemy': 90.00% (72/80)  Epoch 8/10, Loss: 0.4902  📌 Train clase 'enemy': 64.44% (58/90)  📌 Train clase 'n\_enemy': 83.87% (78/93)  Epoch 9/10, Loss: 0.5067  📌 Train clase 'enemy': 47.25% (43/91)  📌 Train clase 'n\_enemy': 94.57% (87/92)  Epoch 10/10, Loss: 0.5171  📌 Train clase 'enemy': 62.22% (56/90)  📌 Train clase 'n\_enemy': 80.65% (75/93)  🔎 Precisión total en test: 69.14%  📂 Clase 'enemy': 56.10% (23/41)  📂 Clase 'n\_enemy': 82.50% (33/40)  Imagen 0: Real = enemy, Predicho = enemy  Imagen 1: Real = enemy, Predicho = n\_enemy  Imagen 2: Real = enemy, Predicho = enemy  Imagen 3: Real = enemy, Predicho = enemy  Imagen 4: Real = enemy, Predicho = enemy  Imagen 5: Real = enemy, Predicho = enemy  Imagen 6: Real = enemy, Predicho = enemy  Imagen 7: Real = enemy, Predicho = enemy  Imagen 8: Real = enemy, Predicho = enemy  Imagen 9: Real = enemy, Predicho = enemy  Imagen 10: Real = enemy, Predicho = enemy  Imagen 11: Real = enemy, Predicho = enemy  Imagen 12: Real = enemy, Predicho = enemy  Imagen 13: Real = enemy, Predicho = n\_enemy  Imagen 14: Real = enemy, Predicho = enemy  Imagen 15: Real = enemy, Predicho = enemy  Imagen 16: Real = enemy, Predicho = enemy  Imagen 17: Real = enemy, Predicho = n\_enemy  Imagen 18: Real = enemy, Predicho = enemy  Imagen 19: Real = enemy, Predicho = n\_enemy  Imagen 20: Real = enemy, Predicho = n\_enemy  Imagen 21: Real = enemy, Predicho = n\_enemy  Imagen 22: Real = enemy, Predicho = n\_enemy  Imagen 23: Real = enemy, Predicho = n\_enemy  Imagen 24: Real = enemy, Predicho = n\_enemy  Imagen 25: Real = enemy, Predicho = n\_enemy  Imagen 26: Real = enemy, Predicho = enemy  Imagen 27: Real = enemy, Predicho = n\_enemy  Imagen 28: Real = enemy, Predicho = n\_enemy  Imagen 29: Real = enemy, Predicho = enemy  Imagen 30: Real = enemy, Predicho = n\_enemy  Imagen 31: Real = enemy, Predicho = enemy  Imagen 32: Real = enemy, Predicho = enemy  Imagen 33: Real = enemy, Predicho = enemy  Imagen 34: Real = enemy, Predicho = enemy  Imagen 35: Real = enemy, Predicho = n\_enemy  Imagen 36: Real = enemy, Predicho = n\_enemy  Imagen 37: Real = enemy, Predicho = n\_enemy  Imagen 38: Real = enemy, Predicho = n\_enemy  Imagen 39: Real = enemy, Predicho = enemy  Imagen 40: Real = enemy, Predicho = n\_enemy  Imagen 41: Real = n\_enemy, Predicho = n\_enemy  Imagen 42: Real = n\_enemy, Predicho = n\_enemy  Imagen 43: Real = n\_enemy, Predicho = n\_enemy  Imagen 44: Real = n\_enemy, Predicho = enemy  Imagen 45: Real = n\_enemy, Predicho = enemy  Imagen 46: Real = n\_enemy, Predicho = n\_enemy  Imagen 47: Real = n\_enemy, Predicho = enemy  Imagen 48: Real = n\_enemy, Predicho = n\_enemy  Imagen 49: Real = n\_enemy, Predicho = n\_enemy  Imagen 50: Real = n\_enemy, Predicho = n\_enemy  Imagen 51: Real = n\_enemy, Predicho = n\_enemy  Imagen 52: Real = n\_enemy, Predicho = n\_enemy  Imagen 53: Real = n\_enemy, Predicho = n\_enemy  Imagen 54: Real = n\_enemy, Predicho = n\_enemy  Imagen 55: Real = n\_enemy, Predicho = n\_enemy  Imagen 56: Real = n\_enemy, Predicho = n\_enemy  Imagen 57: Real = n\_enemy, Predicho = n\_enemy  Imagen 58: Real = n\_enemy, Predicho = n\_enemy  Imagen 59: Real = n\_enemy, Predicho = n\_enemy  Imagen 60: Real = n\_enemy, Predicho = n\_enemy  Imagen 61: Real = n\_enemy, Predicho = n\_enemy  Imagen 62: Real = n\_enemy, Predicho = n\_enemy  Imagen 63: Real = n\_enemy, Predicho = n\_enemy  Imagen 64: Real = n\_enemy, Predicho = n\_enemy  Imagen 65: Real = n\_enemy, Predicho = enemy  Imagen 66: Real = n\_enemy, Predicho = n\_enemy  Imagen 67: Real = n\_enemy, Predicho = enemy  Imagen 68: Real = n\_enemy, Predicho = enemy  Imagen 69: Real = n\_enemy, Predicho = n\_enemy  Imagen 70: Real = n\_enemy, Predicho = n\_enemy  Imagen 71: Real = n\_enemy, Predicho = n\_enemy  Imagen 72: Real = n\_enemy, Predicho = n\_enemy  Imagen 73: Real = n\_enemy, Predicho = n\_enemy  Imagen 74: Real = n\_enemy, Predicho = n\_enemy  Imagen 75: Real = n\_enemy, Predicho = n\_enemy  Imagen 76: Real = n\_enemy, Predicho = enemy  Imagen 77: Real = n\_enemy, Predicho = n\_enemy  Imagen 78: Real = n\_enemy, Predicho = n\_enemy  Imagen 79: Real = n\_enemy, Predicho = n\_enemy  Imagen 80: Real = n\_enemy, Predicho = n\_enemy  📊 Reporte de clasificación por clase:  precision recall f1-score support  enemy 0.77 0.56 0.65 41  n\_enemy 0.65 0.82 0.73 40  accuracy 0.69 81  macro avg 0.71 0.69 0.69 81  weighted avg 0.71 0.69 0.69 81  📌 Matriz de confusión:  [[23 18]  [ 7 33]] |
| Resultado de la ejecución.  Como resultado de esta ejecución vamos a estar observando que muchas de las imágenes se clasificaron de forma incorrecta en el estado de enemy, de acuerdo a las impresiones que estamos observando y de acuerdo a los índices que estuvimos imprimiendo, Relativos a la aproximación en cada una de las épocas podemos concluir que se requiere un poco más de imágenes dentro del conjunto para lo cual procedemos a incrementar nuestro grupo de imágenes.  Así mismo podemos notar en la matriz de confusión en la cual la diagonal principal se mantuvo aunque con índices bajos pero con 23 aciertos del lado de enemy y con 33 en el lado contrario sin embargo también podemos observar que hubo cierto margen de error, en el cual podemos observar 7 errores del lado de la clase enemy y prácticamente 18 sobre la clase contraria |
| Ejecucion 2 |
| Para esta segunda ejecución incrementamos la cantidad de imágenes de nuestro conjunto y editamos algunas para mejorar la protección del patrón |
| C:\Users\lilia\OneDrive\Escritorio\ahl\Mai\redc\_CNN>py CNN.py  Train clases: ['enemy', 'n\_enemy']  Test clases: ['enemy', 'n\_enemy']  Epoch 1/10, Loss: 0.9333  📌 Train clase 'enemy': 35.56% (32/90)  📌 Train clase 'n\_enemy': 65.59% (61/93)  Epoch 2/10, Loss: 0.6915  📌 Train clase 'enemy': 49.40% (41/83)  📌 Train clase 'n\_enemy': 53.00% (53/100)  Epoch 3/10, Loss: 0.6747  📌 Train clase 'enemy': 22.73% (20/88)  📌 Train clase 'n\_enemy': 100.00% (95/95)  Epoch 4/10, Loss: 0.5827  📌 Train clase 'enemy': 40.85% (29/71)  📌 Train clase 'n\_enemy': 91.96% (103/112)  Epoch 5/10, Loss: 0.6162  📌 Train clase 'enemy': 54.74% (52/95)  📌 Train clase 'n\_enemy': 72.73% (64/88)  Epoch 6/10, Loss: 0.6198  📌 Train clase 'enemy': 33.33% (29/87)  📌 Train clase 'n\_enemy': 95.83% (92/96)  Epoch 7/10, Loss: 0.5728  📌 Train clase 'enemy': 68.27% (71/104)  📌 Train clase 'n\_enemy': 62.03% (49/79)  Epoch 8/10, Loss: 0.5421  📌 Train clase 'enemy': 65.22% (60/92)  📌 Train clase 'n\_enemy': 67.03% (61/91)  Epoch 9/10, Loss: 0.5341  📌 Train clase 'enemy': 48.89% (44/90)  📌 Train clase 'n\_enemy': 92.47% (86/93)  Epoch 10/10, Loss: 0.4689  📌 Train clase 'enemy': 53.75% (43/80)  📌 Train clase 'n\_enemy': 93.20% (96/103)  🔎 Precisión total en test: 24.69%  📂 Clase 'enemy': 9.76% (4/41)  📂 Clase 'n\_enemy': 40.00% (16/40)  Imagen 0: Real = enemy, Predicho = enemy  Imagen 1: Real = enemy, Predicho = n\_enemy  Imagen 2: Real = enemy, Predicho = n\_enemy  Imagen 3: Real = enemy, Predicho = n\_enemy  Imagen 4: Real = enemy, Predicho = n\_enemy  Imagen 5: Real = enemy, Predicho = n\_enemy  Imagen 6: Real = enemy, Predicho = n\_enemy  Imagen 7: Real = enemy, Predicho = n\_enemy  Imagen 8: Real = enemy, Predicho = n\_enemy  Imagen 9: Real = enemy, Predicho = n\_enemy  Imagen 10: Real = enemy, Predicho = enemy  Imagen 11: Real = enemy, Predicho = enemy  Imagen 12: Real = enemy, Predicho = enemy  Imagen 13: Real = enemy, Predicho = n\_enemy  Imagen 14: Real = enemy, Predicho = n\_enemy  Imagen 15: Real = enemy, Predicho = n\_enemy  Imagen 16: Real = enemy, Predicho = n\_enemy  Imagen 17: Real = enemy, Predicho = n\_enemy  Imagen 18: Real = enemy, Predicho = n\_enemy  Imagen 19: Real = enemy, Predicho = n\_enemy  Imagen 20: Real = enemy, Predicho = n\_enemy  Imagen 21: Real = enemy, Predicho = n\_enemy  Imagen 22: Real = enemy, Predicho = n\_enemy  Imagen 23: Real = enemy, Predicho = n\_enemy  Imagen 24: Real = enemy, Predicho = n\_enemy  Imagen 25: Real = enemy, Predicho = n\_enemy  Imagen 26: Real = enemy, Predicho = n\_enemy  Imagen 27: Real = enemy, Predicho = n\_enemy  Imagen 28: Real = enemy, Predicho = n\_enemy  Imagen 29: Real = enemy, Predicho = n\_enemy  Imagen 30: Real = enemy, Predicho = n\_enemy  Imagen 31: Real = enemy, Predicho = n\_enemy  Imagen 32: Real = enemy, Predicho = n\_enemy  Imagen 33: Real = enemy, Predicho = n\_enemy  Imagen 34: Real = enemy, Predicho = n\_enemy  Imagen 35: Real = enemy, Predicho = n\_enemy  Imagen 36: Real = enemy, Predicho = n\_enemy  Imagen 37: Real = enemy, Predicho = n\_enemy  Imagen 38: Real = enemy, Predicho = n\_enemy  Imagen 39: Real = enemy, Predicho = n\_enemy  Imagen 40: Real = enemy, Predicho = n\_enemy  Imagen 41: Real = n\_enemy, Predicho = n\_enemy  Imagen 42: Real = n\_enemy, Predicho = n\_enemy  Imagen 43: Real = n\_enemy, Predicho = n\_enemy  Imagen 44: Real = n\_enemy, Predicho = n\_enemy  Imagen 45: Real = n\_enemy, Predicho = n\_enemy  Imagen 46: Real = n\_enemy, Predicho = n\_enemy  Imagen 47: Real = n\_enemy, Predicho = enemy  Imagen 48: Real = n\_enemy, Predicho = n\_enemy  Imagen 49: Real = n\_enemy, Predicho = n\_enemy  Imagen 50: Real = n\_enemy, Predicho = enemy  Imagen 51: Real = n\_enemy, Predicho = n\_enemy  Imagen 52: Real = n\_enemy, Predicho = n\_enemy  Imagen 53: Real = n\_enemy, Predicho = n\_enemy  Imagen 54: Real = n\_enemy, Predicho = enemy  Imagen 55: Real = n\_enemy, Predicho = n\_enemy  Imagen 56: Real = n\_enemy, Predicho = n\_enemy  Imagen 57: Real = n\_enemy, Predicho = enemy  Imagen 58: Real = n\_enemy, Predicho = enemy  Imagen 59: Real = n\_enemy, Predicho = enemy  Imagen 60: Real = n\_enemy, Predicho = n\_enemy  Imagen 61: Real = n\_enemy, Predicho = n\_enemy  Imagen 62: Real = n\_enemy, Predicho = n\_enemy  Imagen 63: Real = n\_enemy, Predicho = enemy  Imagen 64: Real = n\_enemy, Predicho = enemy  Imagen 65: Real = n\_enemy, Predicho = enemy  Imagen 66: Real = n\_enemy, Predicho = enemy  Imagen 67: Real = n\_enemy, Predicho = enemy  Imagen 68: Real = n\_enemy, Predicho = enemy  Imagen 69: Real = n\_enemy, Predicho = enemy  Imagen 70: Real = n\_enemy, Predicho = enemy  Imagen 71: Real = n\_enemy, Predicho = enemy  Imagen 72: Real = n\_enemy, Predicho = enemy  Imagen 73: Real = n\_enemy, Predicho = enemy  Imagen 74: Real = n\_enemy, Predicho = enemy  Imagen 75: Real = n\_enemy, Predicho = enemy  Imagen 76: Real = n\_enemy, Predicho = enemy  Imagen 77: Real = n\_enemy, Predicho = enemy  Imagen 78: Real = n\_enemy, Predicho = enemy  Imagen 79: Real = n\_enemy, Predicho = enemy  Imagen 80: Real = n\_enemy, Predicho = enemy  📊 Reporte de clasificación por clase:  precision recall f1-score support  enemy 0.14 0.10 0.12 41  n\_enemy 0.30 0.40 0.34 40  accuracy 0.25 81  macro avg 0.22 0.25 0.23 81  weighted avg 0.22 0.25 0.23 81  📌 Matriz de confusión:  [[ 4 37]  [24 16]] |
| Resultados  En este caso vemos cómo se mejora un poco la detección sin embargo ya en el conjunto total se empeora esto Puede ser debido a que la cantidad de generaciones es muy baja respecto a lo que queremos obtener podemos verlo con los índices de aproximación |
| Ejecucion 3 |
| Para el caso de la tercera ejecución mantuvimos la cantidad de generaciones durante las cuales se ejecuta el algoritmo sin embargo variamos un poco la cantidad de imágenes incluso aquí se empezaron a editar algunas imágenes de la clase n\_enemy para añadir variedad al patron |
| C:\Users\lilia\OneDrive\Escritorio\ahl\Mai\redc\_CNN>py CNN.py  Train clases: ['enemy', 'n\_enemy']  Test clases: ['enemy', 'n\_enemy']  Epoch 1/10, Loss: 0.7493  📌 Train clase 'enemy': 44.90% (44/98)  📌 Train clase 'n\_enemy': 57.65% (49/85)  Epoch 2/10, Loss: 0.6214  📌 Train clase 'enemy': 56.12% (55/98)  📌 Train clase 'n\_enemy': 63.53% (54/85)  Epoch 3/10, Loss: 0.6431  📌 Train clase 'enemy': 57.29% (55/96)  📌 Train clase 'n\_enemy': 49.43% (43/87)  Epoch 4/10, Loss: 0.5724  📌 Train clase 'enemy': 35.71% (30/84)  📌 Train clase 'n\_enemy': 97.98% (97/99)  Epoch 5/10, Loss: 0.5292  📌 Train clase 'enemy': 34.52% (29/84)  📌 Train clase 'n\_enemy': 97.98% (97/99)  Epoch 6/10, Loss: 0.5103  📌 Train clase 'enemy': 63.16% (60/95)  📌 Train clase 'n\_enemy': 75.00% (66/88)  Epoch 7/10, Loss: 0.5116  📌 Train clase 'enemy': 59.00% (59/100)  📌 Train clase 'n\_enemy': 84.34% (70/83)  Epoch 8/10, Loss: 0.4257  📌 Train clase 'enemy': 79.59% (78/98)  📌 Train clase 'n\_enemy': 61.18% (52/85)  Epoch 9/10, Loss: 0.3971  📌 Train clase 'enemy': 72.41% (63/87)  📌 Train clase 'n\_enemy': 82.29% (79/96)  Epoch 10/10, Loss: 0.4057  📌 Train clase 'enemy': 67.71% (65/96)  📌 Train clase 'n\_enemy': 82.76% (72/87)  🔎 Precisión total en test: 66.67%  📂 Clase 'enemy': 92.68% (38/41)  📂 Clase 'n\_enemy': 40.00% (16/40)  Imagen 0: Real = enemy, Predicho = enemy  Imagen 1: Real = enemy, Predicho = enemy  Imagen 2: Real = enemy, Predicho = enemy  Imagen 3: Real = enemy, Predicho = enemy  Imagen 4: Real = enemy, Predicho = enemy  Imagen 5: Real = enemy, Predicho = enemy  Imagen 6: Real = enemy, Predicho = enemy  Imagen 7: Real = enemy, Predicho = enemy  Imagen 8: Real = enemy, Predicho = enemy  Imagen 9: Real = enemy, Predicho = enemy  Imagen 10: Real = enemy, Predicho = enemy  Imagen 11: Real = enemy, Predicho = enemy  Imagen 12: Real = enemy, Predicho = enemy  Imagen 13: Real = enemy, Predicho = n\_enemy  Imagen 14: Real = enemy, Predicho = enemy  Imagen 15: Real = enemy, Predicho = enemy  Imagen 16: Real = enemy, Predicho = enemy  Imagen 17: Real = enemy, Predicho = n\_enemy  Imagen 18: Real = enemy, Predicho = enemy  Imagen 19: Real = enemy, Predicho = enemy  Imagen 20: Real = enemy, Predicho = enemy  Imagen 21: Real = enemy, Predicho = enemy  Imagen 22: Real = enemy, Predicho = enemy  Imagen 23: Real = enemy, Predicho = enemy  Imagen 24: Real = enemy, Predicho = n\_enemy  Imagen 25: Real = enemy, Predicho = enemy  Imagen 26: Real = enemy, Predicho = enemy  Imagen 27: Real = enemy, Predicho = enemy  Imagen 28: Real = enemy, Predicho = enemy  Imagen 29: Real = enemy, Predicho = enemy  Imagen 30: Real = enemy, Predicho = enemy  Imagen 31: Real = enemy, Predicho = enemy  Imagen 32: Real = enemy, Predicho = enemy  Imagen 33: Real = enemy, Predicho = enemy  Imagen 34: Real = enemy, Predicho = enemy  Imagen 35: Real = enemy, Predicho = enemy  Imagen 36: Real = enemy, Predicho = enemy  Imagen 37: Real = enemy, Predicho = enemy  Imagen 38: Real = enemy, Predicho = enemy  Imagen 39: Real = enemy, Predicho = enemy  Imagen 40: Real = enemy, Predicho = enemy  Imagen 41: Real = n\_enemy, Predicho = n\_enemy  Imagen 42: Real = n\_enemy, Predicho = n\_enemy  Imagen 43: Real = n\_enemy, Predicho = enemy  Imagen 44: Real = n\_enemy, Predicho = enemy  Imagen 45: Real = n\_enemy, Predicho = enemy  Imagen 46: Real = n\_enemy, Predicho = n\_enemy  Imagen 47: Real = n\_enemy, Predicho = enemy  Imagen 48: Real = n\_enemy, Predicho = n\_enemy  Imagen 49: Real = n\_enemy, Predicho = n\_enemy  Imagen 50: Real = n\_enemy, Predicho = n\_enemy  Imagen 51: Real = n\_enemy, Predicho = n\_enemy  Imagen 52: Real = n\_enemy, Predicho = n\_enemy  Imagen 53: Real = n\_enemy, Predicho = n\_enemy  Imagen 54: Real = n\_enemy, Predicho = n\_enemy  Imagen 55: Real = n\_enemy, Predicho = n\_enemy  Imagen 56: Real = n\_enemy, Predicho = n\_enemy  Imagen 57: Real = n\_enemy, Predicho = n\_enemy  Imagen 58: Real = n\_enemy, Predicho = n\_enemy  Imagen 59: Real = n\_enemy, Predicho = n\_enemy  Imagen 60: Real = n\_enemy, Predicho = enemy  Imagen 61: Real = n\_enemy, Predicho = enemy  Imagen 62: Real = n\_enemy, Predicho = n\_enemy  Imagen 63: Real = n\_enemy, Predicho = enemy  Imagen 64: Real = n\_enemy, Predicho = enemy  Imagen 65: Real = n\_enemy, Predicho = enemy  Imagen 66: Real = n\_enemy, Predicho = enemy  Imagen 67: Real = n\_enemy, Predicho = enemy  Imagen 68: Real = n\_enemy, Predicho = enemy  Imagen 69: Real = n\_enemy, Predicho = enemy  Imagen 70: Real = n\_enemy, Predicho = enemy  Imagen 71: Real = n\_enemy, Predicho = enemy  Imagen 72: Real = n\_enemy, Predicho = enemy  Imagen 73: Real = n\_enemy, Predicho = enemy  Imagen 74: Real = n\_enemy, Predicho = enemy  Imagen 75: Real = n\_enemy, Predicho = enemy  Imagen 76: Real = n\_enemy, Predicho = enemy  Imagen 77: Real = n\_enemy, Predicho = enemy  Imagen 78: Real = n\_enemy, Predicho = enemy  Imagen 79: Real = n\_enemy, Predicho = enemy  Imagen 80: Real = n\_enemy, Predicho = enemy  📊 Reporte de clasificación por clase:  precision recall f1-score support  enemy 0.61 0.93 0.74 41  n\_enemy 0.84 0.40 0.54 40  accuracy 0.67 81  macro avg 0.73 0.66 0.64 81  weighted avg 0.73 0.67 0.64 81  📌 Matriz de confusión:  [[38 3]  [24 16]] |
| Resultados  Como podemos observar aquí nuestro resultado mejoró un poco sin embargo podemos seguir detectando esa barrera que se presenta en cuanto a la aproximación del 50% en el apartado de n\_enemy, debido a lo bajo que se nos muestra en la matriz de confusión la cantidad de elementos que se están clasificando de forma errónea vamos a proceder a elevar la cantidad de generaciones durante las que se entrena nuestro algoritmo |
| Ejecucion 4 |
| Durante algunas ejecuciones intermedias se estuvieron haciendo pruebas y detectamos que nuestro número ideal de generar cuáles son las que se va a estar ejecutando el algoritmo va a ser alrededor de 22 épocas vamos a observar una aproximación superior en este caso |
| C:\Users\lilia\OneDrive\Escritorio\ahl\Mai\redc\_CNN>py CNN.py  Train clases: ['enemy', 'n\_enemy']  Test clases: ['enemy', 'n\_enemy']  Epoch 1/22, Loss: 0.9166  📌 Train clase 'enemy': 46.94% (46/98)  📌 Train clase 'n\_enemy': 55.29% (47/85)  Epoch 2/22, Loss: 0.6875  📌 Train clase 'enemy': 84.44% (76/90)  📌 Train clase 'n\_enemy': 30.11% (28/93)  Epoch 3/22, Loss: 0.6682  📌 Train clase 'enemy': 48.28% (42/87)  📌 Train clase 'n\_enemy': 71.88% (69/96)  Epoch 4/22, Loss: 0.5970  📌 Train clase 'enemy': 40.86% (38/93)  📌 Train clase 'n\_enemy': 92.22% (83/90)  Epoch 5/22, Loss: 0.5639  📌 Train clase 'enemy': 36.78% (32/87)  📌 Train clase 'n\_enemy': 98.96% (95/96)  Epoch 6/22, Loss: 0.5684  📌 Train clase 'enemy': 32.93% (27/82)  📌 Train clase 'n\_enemy': 95.05% (96/101)  Epoch 7/22, Loss: 0.5527  📌 Train clase 'enemy': 45.35% (39/86)  📌 Train clase 'n\_enemy': 95.88% (93/97)  Epoch 8/22, Loss: 0.4501  📌 Train clase 'enemy': 64.84% (59/91)  📌 Train clase 'n\_enemy': 85.87% (79/92)  Epoch 9/22, Loss: 0.5056  📌 Train clase 'enemy': 42.86% (36/84)  📌 Train clase 'n\_enemy': 94.95% (94/99)  Epoch 10/22, Loss: 0.4487  📌 Train clase 'enemy': 75.51% (74/98)  📌 Train clase 'n\_enemy': 81.18% (69/85)  Epoch 11/22, Loss: 0.5029  📌 Train clase 'enemy': 70.65% (65/92)  📌 Train clase 'n\_enemy': 75.82% (69/91)  Epoch 12/22, Loss: 0.4789  📌 Train clase 'enemy': 50.60% (42/83)  📌 Train clase 'n\_enemy': 90.00% (90/100)  Epoch 13/22, Loss: 0.3842  📌 Train clase 'enemy': 80.20% (81/101)  📌 Train clase 'n\_enemy': 81.71% (67/82)  Epoch 14/22, Loss: 0.3680  📌 Train clase 'enemy': 81.52% (75/92)  📌 Train clase 'n\_enemy': 86.81% (79/91)  Epoch 15/22, Loss: 0.4216  📌 Train clase 'enemy': 61.96% (57/92)  📌 Train clase 'n\_enemy': 94.51% (86/91)  Epoch 16/22, Loss: 0.4293  📌 Train clase 'enemy': 75.29% (64/85)  📌 Train clase 'n\_enemy': 77.55% (76/98)  Epoch 17/22, Loss: 0.4499  📌 Train clase 'enemy': 64.63% (53/82)  📌 Train clase 'n\_enemy': 93.07% (94/101)  Epoch 18/22, Loss: 0.3869  📌 Train clase 'enemy': 68.00% (68/100)  📌 Train clase 'n\_enemy': 97.59% (81/83)  Epoch 19/22, Loss: 0.4066  📌 Train clase 'enemy': 83.17% (84/101)  📌 Train clase 'n\_enemy': 59.76% (49/82)  Epoch 20/22, Loss: 0.3403  📌 Train clase 'enemy': 73.03% (65/89)  📌 Train clase 'n\_enemy': 93.62% (88/94)  Epoch 21/22, Loss: 0.3208  📌 Train clase 'enemy': 72.15% (57/79)  📌 Train clase 'n\_enemy': 88.46% (92/104)  Epoch 22/22, Loss: 0.2211  📌 Train clase 'enemy': 88.12% (89/101)  📌 Train clase 'n\_enemy': 90.24% (74/82)  🔎 Precisión total en test: 91.36%  📂 Clase 'enemy': 100.00% (41/41)  📂 Clase 'n\_enemy': 82.50% (33/40)  Imagen 0: Real = enemy, Predicho = enemy  Imagen 1: Real = enemy, Predicho = enemy  Imagen 2: Real = enemy, Predicho = enemy  Imagen 3: Real = enemy, Predicho = enemy  Imagen 4: Real = enemy, Predicho = enemy  Imagen 5: Real = enemy, Predicho = enemy  Imagen 6: Real = enemy, Predicho = enemy  Imagen 7: Real = enemy, Predicho = enemy  Imagen 8: Real = enemy, Predicho = enemy  Imagen 9: Real = enemy, Predicho = enemy  Imagen 10: Real = enemy, Predicho = enemy  Imagen 11: Real = enemy, Predicho = enemy  Imagen 12: Real = enemy, Predicho = enemy  Imagen 13: Real = enemy, Predicho = enemy  Imagen 14: Real = enemy, Predicho = enemy  Imagen 15: Real = enemy, Predicho = enemy  Imagen 16: Real = enemy, Predicho = enemy  Imagen 17: Real = enemy, Predicho = enemy  Imagen 18: Real = enemy, Predicho = enemy  Imagen 19: Real = enemy, Predicho = enemy  Imagen 20: Real = enemy, Predicho = enemy  Imagen 21: Real = enemy, Predicho = enemy  Imagen 22: Real = enemy, Predicho = enemy  Imagen 23: Real = enemy, Predicho = enemy  Imagen 24: Real = enemy, Predicho = enemy  Imagen 25: Real = enemy, Predicho = enemy  Imagen 26: Real = enemy, Predicho = enemy  Imagen 27: Real = enemy, Predicho = enemy  Imagen 28: Real = enemy, Predicho = enemy  Imagen 29: Real = enemy, Predicho = enemy  Imagen 30: Real = enemy, Predicho = enemy  Imagen 31: Real = enemy, Predicho = enemy  Imagen 32: Real = enemy, Predicho = enemy  Imagen 33: Real = enemy, Predicho = enemy  Imagen 34: Real = enemy, Predicho = enemy  Imagen 35: Real = enemy, Predicho = enemy  Imagen 36: Real = enemy, Predicho = enemy  Imagen 37: Real = enemy, Predicho = enemy  Imagen 38: Real = enemy, Predicho = enemy  Imagen 39: Real = enemy, Predicho = enemy  Imagen 40: Real = enemy, Predicho = enemy  Imagen 41: Real = n\_enemy, Predicho = enemy  Imagen 42: Real = n\_enemy, Predicho = enemy  Imagen 43: Real = n\_enemy, Predicho = n\_enemy  Imagen 44: Real = n\_enemy, Predicho = n\_enemy  Imagen 45: Real = n\_enemy, Predicho = n\_enemy  Imagen 46: Real = n\_enemy, Predicho = n\_enemy  Imagen 47: Real = n\_enemy, Predicho = n\_enemy  Imagen 48: Real = n\_enemy, Predicho = n\_enemy  Imagen 49: Real = n\_enemy, Predicho = n\_enemy  Imagen 50: Real = n\_enemy, Predicho = n\_enemy  Imagen 51: Real = n\_enemy, Predicho = n\_enemy  Imagen 52: Real = n\_enemy, Predicho = n\_enemy  Imagen 53: Real = n\_enemy, Predicho = n\_enemy  Imagen 54: Real = n\_enemy, Predicho = n\_enemy  Imagen 55: Real = n\_enemy, Predicho = n\_enemy  Imagen 56: Real = n\_enemy, Predicho = n\_enemy  Imagen 57: Real = n\_enemy, Predicho = n\_enemy  Imagen 58: Real = n\_enemy, Predicho = n\_enemy  Imagen 59: Real = n\_enemy, Predicho = n\_enemy  Imagen 60: Real = n\_enemy, Predicho = n\_enemy  Imagen 61: Real = n\_enemy, Predicho = n\_enemy  Imagen 62: Real = n\_enemy, Predicho = n\_enemy  Imagen 63: Real = n\_enemy, Predicho = n\_enemy  Imagen 64: Real = n\_enemy, Predicho = n\_enemy  Imagen 65: Real = n\_enemy, Predicho = n\_enemy  Imagen 66: Real = n\_enemy, Predicho = n\_enemy  Imagen 67: Real = n\_enemy, Predicho = enemy  Imagen 68: Real = n\_enemy, Predicho = enemy  Imagen 69: Real = n\_enemy, Predicho = n\_enemy  Imagen 70: Real = n\_enemy, Predicho = n\_enemy  Imagen 71: Real = n\_enemy, Predicho = n\_enemy  Imagen 72: Real = n\_enemy, Predicho = n\_enemy  Imagen 73: Real = n\_enemy, Predicho = n\_enemy  Imagen 74: Real = n\_enemy, Predicho = enemy  Imagen 75: Real = n\_enemy, Predicho = n\_enemy  Imagen 76: Real = n\_enemy, Predicho = enemy  Imagen 77: Real = n\_enemy, Predicho = n\_enemy  Imagen 78: Real = n\_enemy, Predicho = n\_enemy  Imagen 79: Real = n\_enemy, Predicho = enemy  Imagen 80: Real = n\_enemy, Predicho = n\_enemy  📊 Reporte de clasificación por clase:  precision recall f1-score support  enemy 0.85 1.00 0.92 41  n\_enemy 1.00 0.82 0.90 40  accuracy 0.91 81  macro avg 0.93 0.91 0.91 81  weighted avg 0.93 0.91 0.91 81  📌 Matriz de confusión:  [[41 0]  [ 7 33]]  C:\Users\lilia\OneDrive\Escritorio\ahl\Mai\redc\_CNN> |
| como pudimos observar en estos casos tuvimos un índice de aproximación cercano al 91.3% asimismo de acuerdo a nuestra matriz de confusión y a nuestros resultados en los índices y pesos nuestra matriz de confusión no tuvo errores en cuanto a clasificar lo que viene siendo nuestros enemigos sin embargo si presentó algunos errores todavía en el apartado de n\_enemy.  Para efectos de esta práctica podemos concluir de esta forma en la cual nuestra red neuronal ya presenta mejoría sustancial podríamos incrementar la cantidad de imágenes y seguir haciendo pruebas de entrenamiento sin embargo el modelo actual ya cubre nuestro requerimiento |

## CONCLUSIÓN

En conclusión, el desarrollo de la red neuronal convolucional (CNN) permitió explorar de manera práctica cómo estas arquitecturas pueden aplicarse a la detección y clasificación de patrones visuales dentro de un entorno específico, en este caso, el reconocimiento de enemigos en un videojuego.

A lo largo de las diferentes ejecuciones, se observó que factores como la cantidad y calidad del dataset, el balance de clases, el número de épocas de entrenamiento y las técnicas de regularización influyen directamente en el desempeño del modelo. Los resultados muestran que, aunque la red logró identificar patrones característicos del enemigo con un nivel aceptable de precisión en algunos escenarios, aún existe un margen de error importante, particularmente en la clasificación de la clase *n\_enemy*.

Estos hallazgos reflejan la necesidad de seguir ajustando parámetros, aumentar la diversidad de datos y considerar técnicas más avanzadas como *data augmentation* más robusta, arquitecturas más profundas o el uso de *transfer learning* para mejorar la generalización.

En síntesis, el proyecto demuestra el potencial de las CNNs para tareas de detección visual en tiempo real y confirma que, con un entrenamiento adecuado y un dataset más amplio y balanceado, es posible alcanzar resultados mucho más sólidos y aplicables a contextos reales dentro del campo de la visión por computadora.

## REFERENCIAS (FORMATO APA)

* Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
* Millington, I., & Funge, J. (2016). *Artificial Intelligence for Games* (2nd ed.). CRC Press.