Sistema de Reconocimiento de Pokémon con Visión Artificial

Documentación técnica

9 de mayo de 2025

Índice

1. Introducción **3**

2. Arquitectura del Sistema **3**
- 2.1. Estructura de archivos . . . 3
- 2.2. Componentes principales . . . 3
  - 2.2.1. Clase PokemonDataset . . . 3
  - 2.2.2. Clase PokemonRecognizer . . . 4

3. Obtención y Procesamiento de Datos **5**
- 3.1. Recopilación de datos . . . 5
- 3.2. Preprocesamiento de imágenes . . . 7

4. Modelo de Aprendizaje Profundo **7**
- 4.1. Arquitectura del modelo . . . 7
- 4.2. Proceso de entrenamiento . . . 8
- 4.3. Análisis del proceso de aprendizaje . . . 9
  - 4.3.1. Función de pérdida (Loss function) . . . 9
  - 4.3.2. Optimizador . . . 9
  - 4.3.3. Ciclo de entrenamiento (Training loop) . . . 10
  - 4.3.4. Métricas de seguimiento . . . 10
- 4.4. Evaluación del modelo . . . 10

5. Predicción y Visualización **11**
- 5.1. Proceso de predicción . . . 11
- 5.2. Interfaz de usuario y visualización . . . 12

6. Detección en Tiempo Real **13**

7. Análisis de Rendimiento y Limitaciones **14**
7.1. Factores que afectan el rendimiento 14
7.1.1. Cantidad y diversidad de datos 14
7.1.2. Arquitectura del modelo 14
7.1.3. Técnicas de aumento de datos 14
7.2. Posibles mejoras 15

8. Conclusiones **15**

9. Referencias **16**

1. Introducción

Este documento presenta una descripción detallada del sistema de reconocimiento de Pokémon basado en visión artificial e inteligencia artificial. El sistema es capaz de identificar distintos Pokémon a partir de imágenes capturadas en tiempo real utilizando la cámara del dispositivo. Para ello, emplea técnicas de aprendizaje profundo (deep learning) con redes neuronales convolucionales (CNN) y la biblioteca PyTorch.

El sistema comprende varios componentes clave:

- Una base de datos de Pokémon descargada desde la PokeAPI
- Un conjunto de datos de imágenes para entrenamiento
- Un modelo de red neuronal basado en ResNet18
- Una interfaz de usuario basada en OpenCV para la captura y análisis de imágenes

2. Arquitectura del Sistema

El sistema se compone principalmente de la clase `PokemonRecognizer`, que gestiona todas las funcionalidades principales, y de la clase auxiliar `PokemonDataset`, que implementa la interfaz necesaria para el manejo de los datos de entrenamiento.

2.1. Estructura de archivos

El sistema utiliza la siguiente estructura de directorios:

- `pokemon_data/`: Directorio principal para almacenar todos los datos
- `pokemon_data/images/`: Almacena las imágenes de los Pokémon
- `pokemon_data/pokemon_database.csv`: Base de datos con información de los Pokémon
- `pokemon_data/pokemon_model.pth`: Modelo entrenado guardado
- `pokemon_data/class_mapping.pth`: Mapeo entre índices y nombres de Pokémon

2.2. Componentes principales

2.2.1. Clase PokemonDataset

Esta clase hereda de `torch.utils.data.Dataset` y se encarga de cargar y preprocesar las imágenes de los Pokémon para el entrenamiento y evaluación del modelo. Sus principales características son:

```python
class PokemonDataset(Dataset):
    """Dataset de Pok mon para entrenamiento"""
    def __init__(self, csv_file, img_dir, transform=None):
        self.pokemon_data = pd.read_csv(csv_file)
        self.img_dir = img_dir
        self.transform = transform
        # Crear un mapeo de nombres a  ndices  para entrenamiento
        self.name_to_idx = {name: idx for idx, name in enumerate(self.pokemon_data['name'].unique())}
        self.idx_to_name = {idx: name for name, idx in self.name_to_idx.items()}

    def __len__(self):
        return len(self.pokemon_data)

    def __getitem__(self, idx):
        img_path = self.pokemon_data.iloc[idx]['image_path']
        pokemon_name = self.pokemon_data.iloc[idx]['name']

        # Verificar si la imagen existe
        if os.path.exists(img_path):
            image = Image.open(img_path).convert('RGB')
        else:
            # Imagen por defecto si no existe
            print(f"Advertencia: No se encontr  la imagen {img_path}")
            image = Image.new('RGB', (224, 224), (255, 255, 255))

        if self.transform:
            image = self.transform(image)

        # Usar el mapeo para obtener el  ndice  de clase
        label = self.name_to_idx[pokemon_name]
        return image, label
```

Listing 1: Definición de la clase PokemonDataset

Funciones clave:

- `__init__`: Inicializa el dataset y crea mapeos entre nombres de Pokémon e índices de clase
- `__len__`: Devuelve el número total de muestras en el dataset
- `__getitem__`: Carga una imagen específica y devuelve el par (imagen, etiqueta)

2.2.2. Clase PokemonRecognizer

Esta es la clase principal que controla todo el flujo de trabajo del sistema. Sus métodos principales incluyen:

- `__init__`: Inicializa el reconocedor, configurando directorios, cargando datos y el modelo

- `load_from_csv`: Carga la información de los Pokémon desde un archivo CSV existente
- `download_pokemon_database`: Descarga información e imágenes de Pokémon desde la PokeAPI
- `initialize_model`: Inicializa o carga el modelo de red neuronal
- `train_model`: Entrena el modelo con las imágenes de Pokémon
- `evaluate_model`: Evalúa la precisión del modelo entrenado
- `predict_pokemon`: Predice el Pokémon en una imagen dada
- `display_pokemon_info`: Muestra información del Pokémon identificado en pantalla
- `continuous_detection`: Ejecuta la detección en tiempo real en un hilo separado
- `run_camera`: Maneja la interfaz de usuario con OpenCV

3. Obtención y Procesamiento de Datos

3.1. Recopilación de datos

La información de los Pokémon se obtiene automáticamente desde la PokeAPI, una API RESTful que proporciona datos detallados sobre el universo Pokémon. El sistema descarga:

- Información básica: ID, nombre, altura, peso
- Tipos de Pokémon (fuego, agua, etc.)
- Habilidades
- Imágenes oficiales
- Descripciones en español

Este proceso se realiza en el método `download_pokemon_database()`:

```
def download_pokemon_database(self):
    """Descarga informaci n de Pok mon desde PokeAPI y la guarda
    en CSV"""
    csv_data = [['id', 'name', 'height', 'weight', 'types', '
    abilities', 'image_path', 'description']]

    for pokemon_id in range(1, 152):  # Hasta los primeros 151
    Pok mon
        try:
            print(f"Descargando informaci n de Pok mon #{
    pokemon_id}...")
```

```
            response = requests.get(f"https://pokeapi.co/api/v2/pokemon/{pokemon_id}")
            if response.status_code == 200:
                pokemon_data = response.json()
                name = pokemon_data['name']

                # Procesar y guardar datos
                types = [t['type']['name'] for t in pokemon_data['types']]
                abilities = [a['ability']['name'] for a in pokemon_data['abilities']]

                # Guardar imagen
                image_url = pokemon_data['sprites']['other']['official-artwork']['front_default']
                image_path = os.path.join(self.img_dir, f"{pokemon_id}.png")

                # Descargar y guardar imagen
                try:
                    img_response = requests.get(image_url)
                    if img_response.status_code == 200:
                        with open(image_path, 'wb') as img_file:
                            img_file.write(img_response.content)
                        print(f"Imagen guardada: {image_path}")
                    else:
                        image_path = "no_image"
                except Exception as img_error:
                    image_path = "no_image"

                # Obtener descripci n en espa ol
                description = "No hay descripci n disponible"
                try:
                    species_url = pokemon_data['species']['url']
                    species_response = requests.get(species_url)
                    if species_response.status_code == 200:
                        species_data = species_response.json()
                        spanish_entries = [entry for entry in species_data['flavor_text_entries']
                                           if entry['language']['name'] == 'es']
                        if spanish_entries:
                            description = spanish_entries[-1]['flavor_text'].replace('\n', ' ')
                except Exception:
                    pass

                # Guardar en diccionario y CSV
                self.pokedex[name] = {...}  # Almacena los datos
                csv_data.append([...])      # A ade fila al CSV

                # Para no sobrecargar la API
                time.sleep(0.5)

        except Exception as e:
            print(f"Error al descargar Pok mon ID {pokemon_id}: {e}")
```

```
56
57      # Guardar datos en CSV
58      with open(self.csv_path, 'w', newline='', encoding='utf-8') as
        csvfile:
59          writer = csv.writer(csvfile)
60          writer.writerows(csv_data)
```

Listing 2: Método para descargar la base de datos de Pokémon

3.2. Preprocesamiento de imágenes

Las imágenes se preparan para el entrenamiento mediante una serie de transformaciones definidas con las utilidades de PyTorch:

```
1  transform_train = transforms.Compose([
2      transforms.RandomResizedCrop(224),      # Recorte aleatorio con
        redimensionamiento
3      transforms.RandomHorizontalFlip(),      # Volteo horizontal
       aleatorio
4      transforms.RandomRotation(15),          # Rotaci n aleatoria
5      transforms.ColorJitter(brightness=0.1, contrast=0.1),  #
       Ajustes de color
6      transforms.ToTensor(),                  # Conversi n a tensor
7      transforms.Normalize(mean=[0.485, 0.456, 0.406],       #
       Normalizaci n
8                           std=[0.229, 0.224, 0.225]),
9  ])
```

Listing 3: Transformaciones para las imágenes de entrenamiento

Estas transformaciones son fundamentales para aumentar artificialmente el conjunto de datos y mejorar la generalización del modelo, lo que ayuda a prevenir el sobreajuste (overfitting).

4. Modelo de Aprendizaje Profundo

4.1. Arquitectura del modelo

El sistema utiliza como base la arquitectura ResNet18, una red neuronal convolucional de 18 capas con conexiones residuales que facilitan el entrenamiento de redes profundas. ResNet (Residual Network) fue introducida por Microsoft Research y ganó la competición ILSVRC 2015.

```
1  def initialize_model(self):
2      """Carga un modelo existente o inicializa uno nuevo"""
3      try:
4          # Inicializar el modelo base
5          num_classes = len(self.pokedex)
6          self.model = models.resnet18(pretrained=True)
7          self.model.fc = torch.nn.Linear(512, num_classes)
8
9          # Verificar si existe un modelo guardado previamente
10         if os.path.exists(self.model_path):
```

```
            print(f"Cargando modelo entrenado desde {self.
    model_path}...")
            self.model.load_state_dict(torch.load(self.model_path))
            self.model.eval()
        else:
            print("No se encontr  un modelo entrenado. Se usar
    un modelo nuevo.")
            self.model.eval()
    except Exception as e:
        print(f"Error al inicializar el modelo: {e}")
        # Inicializaci n alternativa en caso de error
```

Listing 4: Inicialización del modelo

El código utiliza un modelo preentrenado de ResNet18 y adapta la capa final (fully-connected) para clasificar entre los diferentes Pokémon. Esta técnica de transferencia de aprendizaje (transfer learning) permite aprovechar las características generales aprendidas por el modelo en el conjunto de datos ImageNet y adaptarlas al problema específico de reconocimiento de Pokémon.

4.2. Proceso de entrenamiento

El entrenamiento del modelo se realiza mediante el método `train_model`, que implementa un ciclo de entrenamiento estándar con:

```
def train_model(self, epochs=5):
    """Entrena el modelo con las im genes descargadas"""
    print("\n=== INICIANDO ENTRENAMIENTO DEL MODELO ===")

    # Crear conjunto de datos
    transform_train = transforms.Compose([...])  # Transformaciones

    # Dataset y DataLoader para entrenamiento
    train_dataset = PokemonDataset(
        csv_file=self.csv_path,
        img_dir=self.img_dir,
        transform=transform_train
    )

    # Guardar el mapeo de  ndices  a nombres
    self.idx_to_name = train_dataset.idx_to_name
    torch.save(self.idx_to_name, self.class_mapping_path)

    train_loader = DataLoader(train_dataset, batch_size=8, shuffle=
    True)

    # Preparar modelo para entrenamiento
    self.model.train()
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(self.model.parameters(), lr=0.001)

    # Ciclo de entrenamiento
    for epoch in range(epochs):
        running_loss = 0.0
        correct = 0
        total = 0
```

```
31
32          print(f"\n poca {epoch+1}/{epochs}")
33
34          for i, (inputs, labels) in enumerate(train_loader):
35              # Entrenar
36              optimizer.zero_grad()
37              outputs = self.model(inputs)
38              loss = criterion(outputs, labels)
39              loss.backward()
40              optimizer.step()
41
42              # Estad sticas
43              running_loss += loss.item()
44              _, predicted = torch.max(outputs.data, 1)
45              total += labels.size(0)
46              correct += (predicted == labels).sum().item()
47
48              # Mostrar progreso
49              if (i+1) % 5 == 0:
50                  print(f"Batch {i+1}/{len(train_loader)} | Loss: {
    running_loss/5:.4f} | Precisi n: {100*correct/total:.2f}%")
51                  running_loss = 0.0
52
53      # Guardar modelo entrenado
54      torch.save(self.model.state_dict(), self.model_path)
55      self.model.eval()
```

Listing 5: Método de entrenamiento del modelo

4.3. Análisis del proceso de aprendizaje

Los aspectos clave del entrenamiento incluyen:

4.3.1. Función de pérdida (Loss function)

El sistema utiliza la `CrossEntropyLoss`, que combina `LogSoftmax` y `NLLLoss` en una sola función. Es la función estándar para problemas de clasificación multiclase:

$$\text{Loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) \tag{1}$$

Esta función penaliza las clasificaciones incorrectas, obligando al modelo a mejorar sus predicciones.

4.3.2. Optimizador

El sistema utiliza el optimizador `Adam` (Adaptive Moment Estimation), un algoritmo de optimización que combina las ventajas de los métodos `AdaGrad` y `RMSProp`. Adam adapta las tasas de aprendizaje de cada parámetro utilizando estimaciones del primer y segundo momento de los gradientes.

4.3.3. Ciclo de entrenamiento (Training loop)

El entrenamiento se realiza en ciclos (epochs), donde:

1. Se recorre todo el conjunto de datos en mini-lotes (batches)
2. Para cada lote:
   a) Se reinician los gradientes a cero
   b) Se realiza la propagación hacia adelante (forward pass)
   c) Se calcula la pérdida
   d) Se realiza la retropropagación (backward pass)
   e) Se actualizan los pesos
3. Se calculan y muestran las métricas (pérdida y precisión)

4.3.4. Métricas de seguimiento

Durante el entrenamiento, se realiza un seguimiento de:

- **Pérdida (Loss)**: Indica cuánto se desvían las predicciones de las etiquetas reales
- **Precisión (Accuracy)**: Porcentaje de clasificaciones correctas

4.4. Evaluación del modelo

Después del entrenamiento, el modelo se evalúa en el método `evaluate_model` para determinar su precisión general:

```
def evaluate_model(self):
    """Eval a el modelo con algunas im genes de prueba"""
    print("\n=== EVALUANDO MODELO ===")

    # Crear dataloader para evaluaci n
    eval_dataset = PokemonDataset(
        csv_file=self.csv_path,
        img_dir=self.img_dir,
        transform=self.transform
    )

    eval_loader = DataLoader(eval_dataset, batch_size=16, shuffle=
    True)

    self.model.eval()
    correct = 0
    total = 0

    # No calcular gradientes durante la evaluaci n
    with torch.no_grad():
        for inputs, labels in eval_loader:
            outputs = self.model(inputs)
            _, predicted = torch.max(outputs.data, 1)
```

```
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"Precisi n del modelo: {accuracy:.2f}%")
```

Listing 6: Método de evaluación del modelo

En un escenario ideal, el conjunto de datos debería dividirse en entrenamiento y prueba, pero en esta implementación se evalúa sobre el mismo conjunto para simplificar.

5. Predicción y Visualización

5.1. Proceso de predicción

Para reconocer un Pokémon en una imagen, el sistema:

```
def predict_pokemon(self, image):
    """Identifica el Pok mon en la imagen usando el modelo entrenado"""
    # Verificar si el modelo existe
    if self.model is None:
        print("Error: El modelo no est  inicializado")
        return None, 0

    # Convertir imagen de OpenCV a formato PIL
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    pil_image = Image.fromarray(image_rgb)

    # Preprocesar la imagen
    input_tensor = self.transform(pil_image)
    input_batch = input_tensor.unsqueeze(0)

    # Hacer predicci n
    self.model.eval()
    with torch.no_grad():
        output = self.model(input_batch)
        probabilities = torch.nn.functional.softmax(output, dim=1)[0]
        prediction_idx = torch.argmax(output, 1).item()
        confidence = probabilities[prediction_idx].item() * 100

    # Obtener el nombre del Pok mon usando el mapeo
    try:
        pokemon_name = self.idx_to_name[prediction_idx]

        # Devolver la informaci n del Pok mon
        if pokemon_name in self.pokedex:
            return self.pokedex[pokemon_name], confidence
        else:
            return None, 0
    except KeyError:
        return None, 0
```

Listing 7: Método de predicción

El proceso incluye:

1. Preprocesamiento de la imagen similar al usado en entrenamiento
2. Aplicación del modelo en modo evaluación
3. Obtención de probabilidades mediante Softmax
4. Selección de la clase con mayor probabilidad
5. Mapeo del índice de clase al nombre del Pokémon
6. Recuperación de información detallada del Pokémon

5.2. Interfaz de usuario y visualización

El sistema proporciona una interfaz de usuario con OpenCV que permite:

- Capturar imágenes con la cámara
- Reconocer Pokémon en tiempo real
- Mostrar información detallada del Pokémon identificado
- Entrenar y evaluar el modelo bajo demanda

La visualización de resultados se realiza mediante el método `display_pokemon_info`:

```
def display_pokemon_info(self, frame, pokemon_info, confidence=None):
    """Muestra la informaci n del Pok mon sobre el frame de video"""
    if not pokemon_info:
        cv2.putText(frame, "No se pudo identificar el Pok mon", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        return frame

    # Crear una copia del frame para no modificar el original
    display_frame = frame.copy()

    # Informaci n general en la parte superior
    name_text = f"{pokemon_info['name'].upper()} (#{pokemon_info['id']})"
    cv2.putText(display_frame, name_text, (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

    if confidence is not None:
        conf_text = f"Confianza: {confidence:.2f}%"
        cv2.putText(display_frame, conf_text, (10, 60),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # Crear un rect ngulo semitransparente para el resto de la informaci n
    h, w = display_frame.shape[:2]
    overlay = display_frame.copy()
```

```
    cv2.rectangle(overlay, (0, h-220), (w, h), (0, 0, 0), -1)
    cv2.addWeighted(overlay, 0.7, display_frame, 0.3, 0,
    display_frame)

    # A adir informaci n detallada
    y_pos = h - 190
    infos = [
        f"Tipos: {', '.join(pokemon_info['types'])}",
        f"Altura: {pokemon_info['height']} m",
        f"Peso: {pokemon_info['weight']} kg",
        f"Habilidades: {', '.join(pokemon_info['abilities'])}"
    ]

    for info in infos:
        cv2.putText(display_frame, info, (10, y_pos),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255),
     1)
        y_pos += 30

    # Mostrar descripci n (partida en l neas si es larga)
    description = pokemon_info.get('description', 'No hay
    descripci n disponible')

    # C digo para mostrar la descripci n dividida en l neas...

    # Mostrar imagen del Pok mon en una esquina
    image_path = pokemon_info['image_path']
    if os.path.exists(image_path):
        # C digo para mostrar la imagen del Pok mon...

    return display_frame
```

Listing 8: Método para mostrar información del Pokémon identificado

6. Detección en Tiempo Real

La detección en tiempo real se implementa mediante un hilo separado que ejecuta predicciones a intervalos regulares:

```
def continuous_detection(self, cap):
    """Ejecuta la detecci n continua de Pok mon en un hilo
    separado"""
    print("Deteccion en tiempo real activada")

    while self.detection_active and cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            print("Error: No se pudo obtener frame")
            self.detection_active = False
            break

        # Verificar si ha pasado suficiente tiempo desde la  ltima
     predicci n
        current_time = time.time()
        if current_time - self.last_prediction_time >= self.
    prediction_interval:
```

```
            # Realizar predicci n
            pokemon_info, confidence = self.predict_pokemon(frame)

            # Actualizar variables de estado
            self.current_prediction = pokemon_info
            self.current_confidence = confidence
            self.last_prediction_time = current_time

    print("Detecci n en tiempo real desactivada")
```

Listing 9: Método para detección continua

Este enfoque permite realizar predicciones a intervalos regulares (por defecto cada 0.5 segundos) sin bloquear la interfaz de usuario principal.

7. Análisis de Rendimiento y Limitaciones

7.1. Factores que afectan el rendimiento

El rendimiento del sistema de reconocimiento se ve afectado por varios factores:

7.1.1. Cantidad y diversidad de datos

El código actual descarga y utiliza las imágenes oficiales de los primeros 151 Pokémon. Limitaciones:

- Conjunto de datos pequeño (un solo ángulo por Pokémon)
- Imágenes muy estandarizadas (dibujos oficiales)
- No incluye variaciones (diferentes formas, shiny, etc.)

7.1.2. Arquitectura del modelo

Se utiliza ResNet18 preentrenado, que proporciona un buen equilibrio entre rendimiento y velocidad. Consideraciones:

- ResNet18 tiene suficiente capacidad para este problema
- La transferencia de aprendizaje ayuda a compensar la escasez de datos
- Otras arquitecturas más modernas podrían mejorar los resultados

7.1.3. Técnicas de aumento de datos

El sistema utiliza técnicas básicas de aumento de datos:

- Recortes aleatorios
- Volteos horizontales

- Rotaciones moderadas
- Ajustes de brillo y contraste

Estas técnicas ayudan a mejorar la generalización, pero podrían ser insuficientes para reconocer Pokémon en condiciones reales muy variadas (diferentes ángulos, iluminación, oclusiones parciales, etc.).

7.2. Posibles mejoras

- **Ampliación del conjunto de datos**: Incorporar múltiples imágenes por Pokémon desde diferentes ángulos y contextos
- **Técnicas avanzadas de aumento de datos**: Utilizar técnicas como MixUp, CutMix, o aumento de datos mediante GAN
- **Arquitecturas alternativas**: Probar modelos más recientes como EfficientNet o Vision Transformer (ViT)
- **Fine-tuning más gradual**: Descongelar gradualmente las capas del modelo preentrenado
- **Validación cruzada**: Implementar validación cruzada para evaluación más robusta
- **Detección de objetos**: Integrar algoritmos de detección de objetos (YOLO, SSD) para localizar primero al Pokémon en la imagen

8. Conclusiones

El sistema desarrollado demuestra la aplicación de técnicas de aprendizaje profundo al reconocimiento de Pokémon en imágenes. Utiliza conceptos fundamentales como:

- Redes neuronales convolucionales (CNN)
- Transferencia de aprendizaje
- Aumento de datos
- Entrenamiento supervisado
- Integración de modelos de IA en aplicaciones interactivas

El enfoque adoptado permite reconocer Pokémon en tiempo real con una interfaz de usuario amigable, demostrando cómo los modelos de aprendizaje profundo pueden integrarse en aplicaciones prácticas y divertidas.

La arquitectura modular del sistema facilita futuras mejoras y expansiones, como la incorporación de más Pokémon, mejoras en el modelo o funcionalidades adicionales.

9. Referencias

- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778).
- PyTorch documentation: `https://pytorch.org/docs/stable/index.html`
- PokeAPI: `https://pokeapi.co/api/v2/pokemon/`