UNIVERSIDAD DE GRANADA

Inteligencia Computacional

Reconocimiento Óptico de Caracteres

Autores

Antonio José Muriel Gálvez
Bruno Otero Galadí

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, octubre de 2024

Índice

Introducción

El objetivo de esta práctica es implementar y evaluar distintos tipos de redes neuronales para el reconocimiento óptico de caracteres (OCR) sobre el conjunto de datos MNIST. MNIST contiene imágenes de dígitos del 0 al 9 y sirve de base para evaluar modelos en tareas de clasificación de imágenes. Para esta práctica se han implementado cuatro modelos:

- Red neuronal simple con capa de entrada y salida de tipo softmax.
- Red neuronal multicapa con una capa oculta de 256 unidades logísticas y una capa de salida softmax.
- Red neuronal convolutiva (CNN) con entrenamiento mediante gradiente descendente estocástico.
- Arquitectura de deep learning basada en autoencoders no supervisados para extracción de características, seguida de una capa softmax.

Implementación

Conjunto de Datos y Preparación

Para entrenar nuestro modelo, usamos imágenes y etiquetas almacenadas en archivos binarios. La función `load_images` lee estas imágenes y normaliza los datos dividiendo por 255, asegurando que los valores estén entre 0 y 1. Esto es importante para que el modelo pueda converger más rápido y de forma estable.

```
def load_images(file_path):
    with open(file_path, 'rb') as f:
        images = np.frombuffer(f.read(), dtype=np.uint8,
            offset=16).reshape(-1, 784)
        images = images / 255.0  # Normalización a [0, 1]
    return images
```

Esta función también convierte las imágenes en un formato adecuado para que puedan ser procesadas por la red, dándoles la forma 784 (un vector plano de 28x28 píxeles).

Inicialización de Parámetros

La inicialización de los parámetros es crucial para asegurar que el modelo aprenda de forma efectiva. La función `initialize_parameters_simple` genera los pesos iniciales con valores pequeños al azar, multiplicados por 0.01 para evitar valores demasiado grandes, y sesgos inicializados en cero.

```
def initialize_parameters_simple(input_size, output_size):
    np.random.seed(42)
    W = np.random.randn(input_size, output_size) * 0.01
    b = np.zeros((1, output_size))
    return W, b
```

Red Neuronal Simple

Propagación Hacia Adelante

La función `forward_propagation_simple` toma los datos de entrada X y los multiplica por los pesos W, suma los sesgos b, y aplica la función `softmax` a los resultados. La `softmax` convierte las salidas en probabilidades, necesarias para realizar la clasificación.

```
def forward_propagation_simple(X, W, b):
    Z = np.dot(X, W) + b
    A = softmax(Z)
    return A
```

Esta función de propagación hacia adelante calcula las predicciones actuales de la red, que serán utilizadas para medir el error con las etiquetas verdaderas.

Cálculo de la Pérdida

El error del modelo se mide con la función de pérdida de entropía cruzada. La función `compute_loss_simple` calcula la entropía cruzada entre las probabilidades predichas y las etiquetas reales, proporcionando una medida de qué tan bien está realizando la clasificación.

```
def compute_loss_simple(A, Y):
    m = Y.shape[0]
    log_likelihood = -np.log(A[range(m), Y])
    loss = np.sum(log_likelihood) / m
    return loss
```

La entropía cruzada es efectiva para tareas de clasificación ya que penaliza duramente las predicciones incorrectas, empujando a la red a ajustar sus parámetros para mejorar su precisión.

Propagación Hacia Atrás

Para optimizar los parámetros de la red, usamos la técnica de retropropagación, calculando los gradientes necesarios para actualizar los pesos y sesgos. La función `backward_propagation_simple` computa estos gradientes.

```
def backward_propagation_simple(X, Y, A):
    m = X.shape[0]
    dZ = A
    dZ[range(m), Y] -= 1
    dZ = dZ / m
    dW = np.dot(X.T, dZ)
    db = np.sum(dZ, axis=0, keepdims=True)
    return dW, db
```

Los gradientes obtenidos (`dW` y `db`) indican cómo deben ajustarse los pesos y los sesgos para reducir la pérdida en las próximas iteraciones de entrenamiento.

Actualización de Parámetros

La función `update_parameters_simple` utiliza el gradiente descendente para ajustar los parámetros en función de los gradientes obtenidos en el paso de retropropagación. Aquí se aplica la tasa de aprendizaje para controlar el tamaño de los ajustes.

```
def update_parameters_simple(W, b, dW, db, learning_rate):
    W = W - learning_rate * dW
    b = b - learning_rate * db
    return W, b
```

Con cada ajuste, el modelo se acerca más a un mínimo de la función de pérdida, logrando así una mejor aproximación a la solución óptima.

Entrenamiento del Modelo

La función `train_neural_network_simple` encapsula el proceso de entrenamiento, repitiendo la propagación hacia adelante, cálculo de pérdida, retropropagación y actualización de parámetros durante múltiples épocas.

```
def train_neural_network_simple(X_train, Y_train, epochs,
    learning_rate):
    W, b = initialize_parameters_simple(X_train.shape[1], np
```

```
        .max(Y_train) + 1)
    for epoch in range(epochs):
        A = forward_propagation_simple(X_train, W, b)
        loss = compute_loss_simple(A, Y_train)
        dW, db = backward_propagation_simple(X_train,
            Y_train, A)
        W, b = update_parameters_simple(W, b, dW, db,
            learning_rate)
        if epoch % 100 == 0:
            print(f"Epoch {epoch}, Loss: {loss}")
    return W, b
```

El bucle de entrenamiento permite que el modelo ajuste sus parámetros gradualmente. Imprimir la pérdida cada 100 épocas proporciona una referencia del progreso de la red en la reducción del error.

Evaluación del Modelo

Finalmente, la función `predict_simple` se usa para evaluar el rendimiento de la red después del entrenamiento. Genera las predicciones para un conjunto de datos y calcula la precisión comparando las predicciones con las etiquetas reales.

```
def predict_simple(X, W, b):
    A = forward_propagation_simple(X, W, b)
    predictions = np.argmax(A, axis=1)
    return predictions
```

Esta función permite evaluar la precisión del modelo en un conjunto de datos de prueba, midiendo qué tan bien generaliza la red.

Red Neuronal Multicapa

El diseño e implementación de una red neuronal multicapa (MLP) aplicada a la clasificación de imágenes del conjunto MNIST. Se detallan las operaciones fundamentales, la arquitectura del modelo y el proceso de entrenamiento. La implementación está acompañada de una explicación teórica para entender el funcionamiento interno de la red.

Arquitectura del modelo

El modelo MLP está compuesto por:

- Una **capa de entrada** con 784 neuronas (28x28 píxeles).

- Dos **capas ocultas**, con 128 y 64 neuronas respectivamente.
- Una **capa de salida** con 10 neuronas, correspondiente a las clases de dígitos.
- Funciones de activación ReLU en las capas ocultas y softmax en la capa de salida.

Componentes del modelo

A continuación, se describe cómo funciona el modelo y cómo se implementa cada componente.

Inicialización de pesos y sesgos

Los pesos se inicializan con valores aleatorios siguiendo la técnica de He, que asegura una distribución adecuada para mejorar la convergencia:

$$W^{[l]} \sim \mathcal{N}\left(0, \frac{2}{n_{l-1}}\right)$$

Los sesgos se inicializan a cero. Esto permite una optimización eficiente durante el entrenamiento.

```
weight = np.random.randn(layer_sizes[i + 1], layer_sizes[i]) * np.sqrt(2. /
    layer_sizes[i])
bias = np.zeros(layer_sizes[i + 1])
```

Propagación hacia adelante

La entrada pasa a través de cada capa del modelo, realizando las siguientes operaciones:

1. Cálculo de la preactivación:

   $$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$$

   donde $a^{[l-1]}$ es la salida de la capa anterior.

2. Aplicación de la función de activación:

   $$a^{[l]} = \text{ReLU}(z^{[l]})$$

   En la última capa, se aplica softmax para calcular probabilidades.

```
z = np.dot(self.activations[-1], w.T) + b
activation = relu(z)
```

Retropropagación

La retropropagación ajusta los pesos y sesgos para minimizar la función de pérdida (entropía cruzada):

$$\mathcal{L} = -\frac{1}{m}\sum_{i=1}^{m} \log p(y_i|x_i)$$

La derivada de la pérdida se propaga hacia atrás utilizando la regla de la cadena. En cada capa:

- Se calcula el gradiente de la activación:

$$\delta^{[l]} = \frac{\partial \mathcal{L}}{\partial z^{[l]}} = \delta^{[l+1]} W^{[l+1]} \odot \text{ReLU}'(z^{[l]})$$

- Se calculan los gradientes de los pesos y sesgos:

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = \delta^{[l]} a^{[l-1]}, \quad \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \delta^{[l]}$$

```
d_z = d_output.dot(self.weights[i + 1]) * (self.zs[i] > 0)
dW = np.dot(d_z.T, self.activations[i])
db = np.sum(d_z, axis=0)
```

Actualización con momentum

Se utiliza un optimizador con momentum para acelerar la convergencia:

$$v_{dw} = \beta v_{dw} - \alpha \frac{\partial \mathcal{L}}{\partial W}, \quad W = W + v_{dw}$$

Donde β es el factor de momentum y α es la tasa de aprendizaje.

Entrenamiento y evaluación

El modelo se entrena durante 20 épocas usando mini-lotes de tamaño 32. En cada iteración:

1. Se realiza una propagación hacia adelante.
2. Se calcula la pérdida.
3. Se ajustan los pesos y sesgos mediante retropropagación.

Tras el entrenamiento, el modelo se evalúa calculando la precisión en los conjuntos de entrenamiento y prueba.

Red Neuronal Convolutiva

La red neuronal convolucional (CNN) diseñada para la tarea de clasificación de imágenes del conjunto MNIST. Este modelo está compuesto por capas convolucionales, de pooling, y densas, que trabajan conjuntamente para identificar los dígitos escritos a mano en las imágenes. A continuación, se describen los principios teóricos de las redes convolucionales y el diseño de la arquitectura propuesta.

Arquitectura del modelo

El modelo está compuesto por las siguientes etapas:

1. **Capa convolucional:** Extrae características locales de las imágenes aplicando filtros (kernels). Esta operación aprende patrones como bordes, curvas y texturas en niveles iniciales.
2. **Capa de activación (ReLU):** Introduce no linealidad en el modelo, permitiendo capturar relaciones complejas en los datos.
3. **Capa de pooling (Max-Pooling):** Reduce la dimensionalidad espacial seleccionando el valor máximo en regiones pequeñas de los mapas de características.
4. **Capa totalmente conectada:** Combina las características aprendidas en etapas anteriores para realizar la clasificación final.
5. **Función softmax:** Convierte las salidas en probabilidades normalizadas para cada clase.

Funcionamiento del modelo

El modelo sigue un flujo definido por las operaciones descritas. A continuación, se detallan cada una de estas etapas, con una referencia directa al código adjunto.

Capa convolucional

La operación de convolución se realiza utilizando la función `conv_batch`, que aplica un conjunto de filtros a cada imagen. Durante esta operación:

- Se recorre la imagen usando una ventana del tamaño del filtro.
- Se calcula el producto punto entre el filtro y la región de la imagen.
- Se genera un nuevo mapa de características para cada filtro.

Esta operación permite aprender patrones locales de las imágenes, como bordes y formas. Los filtros se inicializan aleatoriamente y se optimizan durante el entrenamiento.

```
output = np.zeros((num_images, num_filters, out_height,
    out_width))
for i in range(out_height):
    for j in range(out_width):
        region = padded_images[:, :, i*stride:i*stride+
            filter_height, j*stride:j*stride+filter_width]
        output[:, :, i, j] = np.tensordot(region, filters,
            axes=([1, 2, 3], [1, 2, 3]))
```

Capa de activación ReLU

Tras la convolución, se aplica la función de activación **ReLU** (*Rectified Linear Unit*), que se define como:

$$f(x) = \text{máx}(0, x)$$

Esto elimina valores negativos del mapa de características, introduciendo no linealidad y mejorando la capacidad del modelo para aprender patrones complejos.

Capa de pooling

El *Max-Pooling* reduce la dimensionalidad espacial seleccionando el valor máximo en regiones de tamaño fijo, lo que:

- Reduce la cantidad de parámetros del modelo.
- Introduce invariancia a pequeñas traslaciones en la entrada.

```
for i in range(out_height):
    for j in range(out_width):
        region = input_images[:, :, i*stride:i*stride+
            pool_size, j*stride:j*stride+pool_size]
        output[:, :, i, j] = np.max(region, axis=(2, 3))
```

Capa totalmente conectada

La salida de las capas anteriores se aplana y pasa a través de una capa densamente conectada. Esta capa combina todas las características aprendidas y genera una predicción para cada clase.

Función softmax

La salida final se normaliza utilizando la función softmax:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Esto asegura que las salidas sean probabilidades que suman 1.

Entrenamiento

El entrenamiento sigue un esquema de mini-lotes. En cada iteración:

1. Se realiza una **propagación hacia adelante** para calcular las predicciones.
2. Se evalúa la **función de pérdida**, en este caso la entropía cruzada:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \log p(y_i|x_i)$$

3. Se calculan los gradientes mediante retropropagación.
4. Los pesos del modelo se actualizan usando descenso de gradiente.

```
# Retropropagación
d_out = softmax_outs
d_out[np.arange(batch_size), batch_labels] -= 1
d_out /= batch_size

# Gradientes y actualización
dW_dense = np.dot(d_out.T, flat_outs)
db_dense = d_out.sum(axis=0)
dense_weights -= learning_rate * dW_dense
dense_bias -= learning_rate * db_dense
```

Evaluación del modelo

Para evaluar el modelo, se calcula la precisión (*accuracy*) en los conjuntos de entrenamiento y prueba. Esto se realiza comparando las predicciones del modelo con las etiquetas reales:

$$\text{Precisión} = \frac{\text{Número de predicciones correctas}}{\text{Total de muestras}}$$

Red Neuronal Convolucional v2 Mejorado

El modelo de red neuronal convolucional (CNN) presentado incluye optimizaciones significativas respecto al modelo multicapa (MLP) previo, permitiendo

una extracción más eficiente de características y logrando una mayor precisión en tareas como la clasificación de imágenes del conjunto de datos MNIST. A continuación, se describen los principales componentes y mejoras implementadas.

Operación de Convolución Optimizada (im2col)

La operación de convolución ha sido optimizada mediante la función `im2col`, la cual transforma las imágenes y los filtros convolucionales en representaciones matriciales. Esto permite:

- Realizar convoluciones mediante multiplicaciones de matrices, que son operaciones más eficientes en hardware moderno.
- Reducir el costo computacional en comparación con las implementaciones tradicionales basadas en bucles.

Esta optimización resulta en un entrenamiento más rápido y escalable, especialmente cuando se usa hardware especializado como GPUs.

Capas Convolutivas

Las capas convolutivas se utilizan para extraer características locales importantes de las imágenes, preservando la estructura espacial. En este modelo, empleamos filtros de tamaño 3×3, una configuración estándar para capturar patrones básicos como bordes, esquinas y texturas. Estas capas permiten que la red sea más eficiente y robusta frente a transformaciones locales en las imágenes.

Max Pooling

El modelo incluye una capa de *max pooling*, que:

- Reduce las dimensiones de las características extraídas, disminuyendo la complejidad computacional de las capas subsiguientes.
- Retiene las características más relevantes en cada región de la imagen.

Decaimiento de Tasa de Aprendizaje

Para mejorar la convergencia del entrenamiento, se implementó un decaimiento exponencial de la tasa de aprendizaje (*learning rate*). Este enfoque reduce gradualmente la magnitud de los ajustes en los parámetros a medida que se avanza en las épocas, ayudando a evitar oscilaciones o convergencia prematura.

Arquitectura del Modelo

El modelo CNN propuesto sigue la siguiente estructura:

- Entrada: Imágenes de 28×28 píxeles normalizadas.
- Capa Convolutiva: 8 filtros de 3×3 con activación ReLU y un *padding* de 1 píxel para mantener las dimensiones espaciales.
- Capa de Max Pooling: Tamaño de ventana 2×2 con *stride* 2, reduciendo las dimensiones espaciales a la mitad.
- Capa Densa: Totalmente conectada con 10 neuronas (una por cada clase) y una activación *softmax* para la clasificación.

Mejoras Implementadas

Operación de Convolución Optimizada (im2col)

La operación de convolución ha sido optimizada mediante la función `im2col`, la cual transforma las imágenes y los filtros convolucionales en representaciones matriciales. Esto permite:

- Realizar convoluciones mediante multiplicaciones de matrices, que son operaciones más eficientes en hardware moderno.
- Reducir el costo computacional en comparación con las implementaciones tradicionales basadas en bucles.

Esta optimización resulta en un entrenamiento más rápido y escalable, especialmente cuando se usa hardware especializado como GPUs.

Capas Convolutivas

Las capas convolutivas se utilizan para extraer características locales importantes de las imágenes, preservando la estructura espacial. En este modelo, empleamos filtros de tamaño 3×3, una configuración estándar para capturar patrones básicos como bordes, esquinas y texturas. Estas capas permiten que la red sea más eficiente y robusta frente a transformaciones locales en las imágenes.

Max Pooling

El modelo incluye una capa de *max pooling*, que:

- Reduce las dimensiones de las características extraídas, disminuyendo la complejidad computacional de las capas subsiguientes.
- Retiene las características más relevantes en cada región de la imagen.

Decaimiento de Tasa de Aprendizaje

Para mejorar la convergencia del entrenamiento, se implementó un decaimiento exponencial de la tasa de aprendizaje (*learning rate*). Este enfoque reduce gradualmente la magnitud de los ajustes en los parámetros a medida que se avanza en las épocas, ayudando a evitar oscilaciones o convergencia prematura.

Arquitectura del Modelo

El modelo CNN propuesto sigue la siguiente estructura:

- Entrada: Imágenes de 28×28 píxeles normalizadas.
- Capa Convolutiva: 8 filtros de 3×3 con activación ReLU y un *padding* de 1 píxel para mantener las dimensiones espaciales.
- Capa de Max Pooling: Tamaño de ventana 2×2 con *stride* 2, reduciendo las dimensiones espaciales a la mitad.
- Capa Densa: Totalmente conectada con 10 neuronas (una por cada clase) y una activación *softmax* para la clasificación.

Red Neuronal Convoluciona con Librerias

El modelo de red neuronal convolucional (CNN) presentado ha sido diseñado para la tarea de clasificación de imágenes utilizando el conjunto de datos MNIST. Se han aplicado técnicas avanzadas de regularización, optimización y aumento de datos para mejorar el rendimiento del modelo. A continuación, se explica detalladamente la arquitectura y los componentes del modelo.

Data Aumentation

El aumento de datos (Data Augmentation) se utiliza para incrementar la diversidad del conjunto de entrenamiento. Esto ayuda al modelo a generalizar mejor, evitando el sobreajuste. Las transformaciones aplicadas son:

- **Rotación**: Hasta $\pm 15°$.
- **Desplazamiento horizontal y vertical**: Hasta el 10 % del tamaño de la imagen.
- **Zoom**: Hasta un 10 % de aumento o reducción.

```
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1
)
datagen.fit(train_images)
```

Primera capa convolucional

- **32 filtros** de tamaño 3×3.
- Función de activación **ReLU**.
- **Batch Normalization**: Normaliza las activaciones para estabilizar el entrenamiento.
- **MaxPooling**: Reduce las dimensiones espaciales de 28×28 a 14×14 seleccionando el valor máximo en regiones de 2×2.
- **Dropout**: Apaga aleatoriamente el 25 % de las neuronas para evitar sobreajuste.

```
model.add(layers.Conv2D(32, (3, 3), activation='relu',
    input_shape=(28, 28, 1)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))
```

Segunda y tercera capas convolucionales

Segunda capa:

- **64 filtros** de tamaño 3×3.
- Función de activación **ReLU**.
- **Batch Normalization**.
- **MaxPooling**: Reduce las dimensiones espaciales de 14×14 a 7×7.
- **Dropout**: Apaga aleatoriamente el 25 % de las neuronas.

Tercera capa:

- **128 filtros** de tamaño 3×3.
- Función de activación **ReLU**.
- **Batch Normalization**.
- **Flatten**: Convierte la salida tridimensional en un vector plano para conectarse a las capas densas.

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Flatten())
```

Capas densas

- Una capa completamente conectada (**Dense**) con 128 neuronas y función de activación **ReLU**.
- **Batch Normalization**.
- **Dropout**: Apaga aleatoriamente el 50 % de las neuronas.
- Capa de salida con 10 neuronas (para cada clase) y función de activación **Softmax**.
- Regularización **L2** para prevenir el sobreajuste.

```
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))

model.add(layers.Dense(10, activation='softmax',
    kernel_regularizer=regularizers.l2(1e-4)))
```

Optimización y callbacks

- **AdamW**: Variante del optimizador Adam con decaimiento del peso, lo que mejora la regularización.
- **EarlyStopping**: Finaliza el entrenamiento si la pérdida de validación no mejora después de 5 épocas consecutivas.
- **ReduceLROnPlateau**: Reduce la tasa de aprendizaje en un factor de 0.5 si la pérdida de validación no mejora en 3 épocas.

```
model.compile(optimizer=tf.keras.optimizers.AdamW(
    learning_rate=1e-3, weight_decay=1e-4),
              loss='categorical_crossentropy',
```

```
                metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience
    =5, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor
    =0.5, patience=3, min_lr=1e-5)
```

Red Neuronal Convoluciona v2 con Librerias

En este trabajo se presenta un modelo mejorado para la clasificación de imágenes del conjunto de datos MNIST utilizando redes neuronales convolucionales (CNN). El modelo ha sido optimizado a través de varias técnicas avanzadas como regularización, aumento de datos, optimización y callbacks, lo que ha resultado en una mejora significativa del rendimiento.

Arquitectura del Modelo

El modelo se ha construido utilizando una arquitectura profunda con tres bloques convolucionales. Cada bloque consta de capas convolucionales seguidas de normalización por lotes (`BatchNormalization`), activación ReLU y capas de `MaxPooling` para reducir la dimensionalidad.

- **Primer Bloque Convolucional**: Este bloque contiene dos capas convolucionales con 64 filtros. Tras cada capa convolucional, se aplica una normalización por lotes y una capa de `MaxPooling` con un tamaño de 2x2. Además, se aplica `Dropout` con un valor de 0.3 para evitar el sobreajuste.
- **Segundo Bloque Convolucional**: Similar al primero, pero con 128 filtros en las capas convolucionales. Se utiliza la misma técnica de normalización por lotes y `Dropout` con un valor de 0.4.
- **Tercer Bloque Convolucional**: Este bloque contiene 256 filtros en las capas convolucionales y se utiliza `GlobalAveragePooling2D` en lugar de `Flatten` para reducir la dimensionalidad, lo cual mejora la generalización.
- **Capa Densa**: Una capa densa de 256 unidades con activación ReLU, seguida de `BatchNormalization` y `Dropout` con un valor de 0.5 para prevenir el sobreajuste.
- **Capa de Salida**: Finalmente, se utiliza una capa de salida con 10 unidades y activación `softmax` para la clasificación en 10 clases (dígitos del 0 al 9).

El código correspondiente a la definición del modelo es el siguiente:

```python
def build_advanced_model():
    model = models.Sequential()
    model.add(layers.Input(shape=(28, 28, 1)))

    # Primer bloque convolucional
    model.add(layers.Conv2D(64, (3, 3), activation='relu',
        kernel_regularizer=regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(64, (3, 3), activation='relu',
        kernel_regularizer=regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.3))

    # Segundo bloque convolucional
    model.add(layers.Conv2D(128, (3, 3), activation='relu',
        kernel_regularizer=regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(128, (3, 3), activation='relu',
        kernel_regularizer=regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.4))

    # Tercer bloque convolucional
    model.add(layers.Conv2D(256, (3, 3), activation='relu',
        kernel_regularizer=regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.GlobalAveragePooling2D())

    # Capa densa
    model.add(layers.Dense(256, activation='relu',
        kernel_regularizer=regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.5))

    # Capa de salida
    model.add(layers.Dense(10, activation='softmax'))

    return model
```

Regularización

Se ha implementado la técnica de `BatchNormalization` en cada capa convolucional y densa. La normalización por lotes ayuda a estabilizar el entrenamiento y mejorar la convergencia. Esta técnica también contribuye a la reducción del sobreajuste.

Además, se ha utilizado `Dropout` en diferentes partes de la red (0.3 en las capas convolucionales, 0.4 en el segundo bloque y 0.5 en la capa densa) para evitar que el modelo dependa demasiado de ciertas neuronas y, de este modo, mejorar la generalización.

Optimización

El modelo utiliza el optimizador `AdamW`, que es una mejora del optimizador Adam. Este optimizador incorpora decaimiento de peso (L2 regularization) durante las actualizaciones de los parámetros. Esto ayuda a reducir el sobreajuste y mejora la generalización.

El código correspondiente para compilar el modelo es el siguiente:

```
model.compile(optimizer=tf.keras.optimizers.AdamW(
    learning_rate=1e-3, weight_decay=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Callbacks

Se han implementado dos callbacks importantes para mejorar el rendimiento del modelo:

- **EarlyStopping**: Este callback detiene el entrenamiento si la pérdida de validación no mejora después de 7 épocas, lo que evita el sobreentrenamiento. Además, restaura los mejores pesos del modelo durante el entrenamiento.
- **ReduceLROnPlateau**: Este callback reduce la tasa de aprendizaje en un 50 % si la pérdida de validación no mejora después de 3 épocas consecutivas, lo que permite que el modelo se ajuste finamente en las últimas etapas del entrenamiento.

El código correspondiente a los callbacks es:

```
early_stopping = EarlyStopping(monitor='val_loss', patience
    =7, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor
    =0.5, patience=3, min_lr=1e-5)
```

Aumento de Datos

Se ha utilizado un generador de datos (`ImageDataGenerator`) con varias técnicas de aumento de datos, como rotación, desplazamiento y zoom. Esto permite

generar nuevas imágenes a partir de las imágenes originales, lo que mejora la capacidad de generalización del modelo.

El código correspondiente al aumento de datos es:

```
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1
)
datagen.fit(train_images)
```

Evaluación del Modelo

Después de entrenar el modelo, se evalúa en el conjunto de prueba. El rendimiento se mide en términos de precisión (`accuracy`) y pérdida (`loss`). También se generan gráficos que muestran la evolución de la pérdida y la precisión durante el entrenamiento.

El código para evaluar el modelo es:

```
test_loss, test_acc = model.evaluate(test_images,
    test_labels)
print(f'Loss en el conjunto de test: {test_loss}')
print(f'Precisión en el conjunto de test: {test_acc}')
```

Además, se muestra un conjunto de imágenes mal clasificadas para comprender mejor las áreas problemáticas del modelo:

```
predictions = model.predict(test_images)
misclassified_indices = np.where(np.argmax(predictions, axis
    =1) != np.argmax(test_labels, axis=1))[0]

for i in range(min(5, len(misclassified_indices))):
    idx = misclassified_indices[i]
    plt.imshow(test_images[idx].reshape(28, 28), cmap='gray'
        )
    plt.title(f"Real: {np.argmax(test_labels[idx])},
        Predicho: {np.argmax(predictions[idx])}")
    plt.axis('off')
    plt.show()
```

Deep Learning con Autoencoders

Red neuronal profunda diseñado específicamente para la clasificación de imágenes del conjunto de datos MNIST. El modelo consta de tres capas ocultas, cada una con optimizaciones diseñadas para mejorar su desempeño y estabilidad durante el entrenamiento.

Fundamentos del Modelo

El modelo sigue un enfoque supervisado donde las imágenes, inicialmente representadas como matrices de píxeles de 28x28, se convierten en vectores unidimensionales de 784 características. Estas características sirven como entrada para la red neuronal, la cual está diseñada para predecir una de las 10 clases posibles correspondientes a los dígitos del 0 al 9.

Estructura General de la Red

La red neuronal implementada tiene las siguientes características clave:

- **Capa de Entrada:** Acepta vectores de 784 características normalizadas en el rango [0, 1].
- **Capas Ocultas:** Tres capas densamente conectadas (*fully connected*) con 254 neuronas cada una.
- **Capa de Salida:** Una capa con 10 neuronas, correspondiente al número de clases en el problema de clasificación, y una activación *softmax*.

Cada capa oculta utiliza una función de activación *Leaky ReLU*, mientras que la capa de salida emplea una función *softmax* para convertir los valores de salida en probabilidades.

Diseño y Mejoras Implementadas

Inicialización de Pesos

La inicialización de pesos es un factor crítico para asegurar un entrenamiento eficiente y estable en redes neuronales profundas. En este modelo, se emplea el método de inicialización de Xavier (o Glorot), el cual ajusta los valores iniciales de los pesos para que las activaciones mantengan una varianza constante a lo largo de las capas. Esto se calcula como:

$$W \sim \mathcal{N}(0, \frac{2}{\text{fan\_in}})$$

donde *fan_in* es el número de conexiones de entrada a una neurona. Este método reduce significativamente el riesgo de desvanecimiento o explosión de gradientes.

Funciones de Activación

La función de activación seleccionada para las capas ocultas es *Leaky ReLU*, definida como:

$$f(z) = \begin{cases} z & \text{si } z > 0 \\ \alpha z & \text{si } z \leq 0 \end{cases}$$

con un valor típico de $\alpha = 0.01$. A diferencia de la función *ReLU*, *Leaky ReLU* permite que los valores negativos fluyan a través de la red, mitigando el problema de gradientes muertos.

En la capa de salida, se emplea *softmax*, definida como:

$$\sigma(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Esta función garantiza que las salidas sean probabilidades normalizadas, facilitando la interpretación de los resultados.

Regularización con Dropout

Para combatir el sobreajuste, se utiliza *dropout* en las capas ocultas. Este método desactiva aleatoriamente una fracción de neuronas durante cada paso de entrenamiento, mejorando la generalización del modelo.

Propagación hacia Adelante y Retropropagación

Propagación Hacia Adelante: Durante esta etapa, las entradas pasan por cada capa oculta, calculándose las activaciones intermedias y, finalmente, las probabilidades en la capa de salida. Las activaciones son almacenadas para ser utilizadas en la retropropagación.

Retropropagación: En esta etapa, se calculan los gradientes de la función de pérdida respecto a los parámetros del modelo mediante el algoritmo de retropropagación. Los gradientes se ajustan utilizando descenso de gradiente estocástico (SGD).

Hiperparámetros y Configuración

Los hiperparámetros seleccionados para el modelo son los siguientes:

- **Tamaño del Lote:** 64 muestras por mini-lote.
- **Tasa de Aprendizaje:** 0.001, ajustada para proporcionar un equilibrio entre velocidad de convergencia y estabilidad.

- **Número de Épocas:** 220 épocas para garantizar una convergencia adecuada.
- **Tasa de Dropout:** 50 % para las capas ocultas.

Cálculo de la Función de Pérdida

Se utiliza la función de entropía cruzada categórica para medir la discrepancia entre las predicciones del modelo y las etiquetas verdaderas. Está definida como:

$$\mathcal{L} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$$

donde y_{ij} es la etiqueta verdadera (en formato *one-hot*) y $\hat{y}_{ij}$ es la probabilidad predicha para la clase j.

Entrenamiento del Modelo

El modelo es entrenado en el conjunto de entrenamiento, utilizando mini-lotes para mejorar la eficiencia computacional y reducir la varianza en las actualizaciones de los gradientes. En cada época, se evalúa la pérdida en el conjunto de validación para monitorear el progreso y ajustar los hiperparámetros si es necesario.

Evaluación del Desempeño

La evaluación se realiza comparando las etiquetas predichas con las etiquetas verdaderas en el conjunto de prueba. La métrica principal utilizada es la precisión, definida como:

$$\text{Precisión} = \frac{\text{Número de predicciones correctas}}{\text{Número total de muestras}}$$

Deep Learning con Autoencoders v2

Red neuronal profunda diseñada para la clasificación de imágenes del conjunto de datos MNIST. Esta red consta de cuatro capas ocultas totalmente conectadas y está optimizada para maximizar la precisión en la clasificación.

Estructura del Modelo

El diseño de la red incluye los siguientes elementos clave:

- **Capa de Entrada:** Las imágenes, originalmente de tamaño 28×28, se aplanan a vectores de 784 características normalizadas en el rango [0, 1].

- **Capas Ocultas:** Cuatro capas densamente conectadas con 128 neuronas cada una y funciones de activación Leaky ReLU.
- **Capa de Salida:** Una capa totalmente conectada con 10 neuronas que utiliza la función de activación *softmax* para generar probabilidades de las clases.

Inicialización de Pesos

La inicialización de pesos es fundamental para el entrenamiento eficiente de redes neuronales profundas. Se utiliza el método de inicialización de Xavier (o Glorot), donde los pesos son inicializados según:

$$W \sim \mathcal{N}(0, \frac{2}{\text{fan\_in}})$$

Esta técnica asegura que las activaciones mantengan una varianza constante a lo largo de las capas, reduciendo el riesgo de desvanecimiento o explosión de gradientes.

Funciones de Activación

- **Leaky ReLU:** Las capas ocultas emplean la función *Leaky ReLU*, definida como:
$$f(z) = \begin{cases} z & \text{si } z > 0 \\ \alpha z & \text{si } z \leq 0 \end{cases}$$
con $\alpha = 0.01$. Esta función mejora el flujo de gradientes en comparación con ReLU, especialmente en regiones de activaciones negativas.
- **Softmax:** La capa de salida utiliza *softmax*, que convierte las salidas en probabilidades normalizadas:
$$\sigma(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Regularización y Estabilización

Para reducir el sobreajuste y estabilizar el entrenamiento, se implementaron las siguientes técnicas:

- **Dropout:** En las capas ocultas se desactivan aleatoriamente neuronas durante el entrenamiento con una probabilidad $p = 0.5$.
- **Batch Normalization:** Se normalizan las activaciones para mejorar la convergencia y minimizar la sensibilidad a la inicialización de pesos.

Propagación hacia Adelante y Retropropagación

Propagación hacia Adelante: El modelo realiza una secuencia de operaciones a través de las cuatro capas ocultas y la capa de salida, calculando las activaciones intermedias y generando probabilidades al final. La arquitectura general es la siguiente:

$$
\begin{aligned}
Z^{[1]} &= XW^{[1]} + b^{[1]} & A^{[1]} &= \text{Leaky ReLU}(Z^{[1]}) \\
Z^{[2]} &= A^{[1]}W^{[2]} + b^{[2]} & A^{[2]} &= \text{Leaky ReLU}(Z^{[2]}) \\
Z^{[3]} &= A^{[2]}W^{[3]} + b^{[3]} & A^{[3]} &= \text{Leaky ReLU}(Z^{[3]}) \\
Z^{[4]} &= A^{[3]}W^{[4]} + b^{[4]} & A^{[4]} &= \text{Leaky ReLU}(Z^{[4]}) \\
Z^{[5]} &= A^{[4]}W^{[5]} + b^{[5]} & A^{[5]} &= \text{softmax}(Z^{[5]})
\end{aligned}
$$

Retropropagación: El algoritmo de retropropagación calcula los gradientes de la función de pérdida respecto a los pesos utilizando el descenso de gradiente. Para la función de activación Leaky ReLU, la derivada se define como:

$$
f'(z) = \begin{cases} 1 & \text{si } z > 0 \\ \alpha & \text{si } z \leq 0 \end{cases}
$$

Optimización y Hiperparámetros

Los hiperparámetros principales utilizados en el entrenamiento fueron:

- **Tamaño del Lote:** 64 muestras por mini-lote.
- **Tasa de Aprendizaje:** 0.001, utilizando un esquema de disminución gradual.
- **Épocas:** 150 épocas para garantizar una convergencia adecuada.
- **Tasa de Dropout:** $p = 0.5$ para las capas ocultas.

La optimización se realiza mediante descenso de gradiente estocástico (SGD), que ajusta los pesos según:

$$
W^{[l]} := W^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial W^{[l]}}
$$

donde η es la tasa de aprendizaje y $\mathcal{L}$ la función de pérdida.

Función de Pérdida

Se utiliza la entropía cruzada categórica para medir la discrepancia entre las predicciones y las etiquetas verdaderas:

$$
\mathcal{L} = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})
$$

donde m es el número de muestras, C el número de clases, y_{ij} la etiqueta verdadera y $\hat{y}_{ij}$ la probabilidad predicha.

Entrenamiento y Evaluación

El modelo fue entrenado en el conjunto de datos de entrenamiento y validado en el conjunto de prueba. La métrica principal de desempeño fue la precisión, definida como:

$$\text{Precisión} = \frac{\text{Número de predicciones correctas}}{\text{Número total de muestras}}$$

Experimentación y Resultados

Resultados de la Red Neuronal Simple

Se analizan los resultados obtenidos al entrenar una red neuronal simple para la clasificación de dígitos en el conjunto de datos MNIST. La red consta de una capa de entrada, una capa oculta y una capa de salida con activación `softmax`.

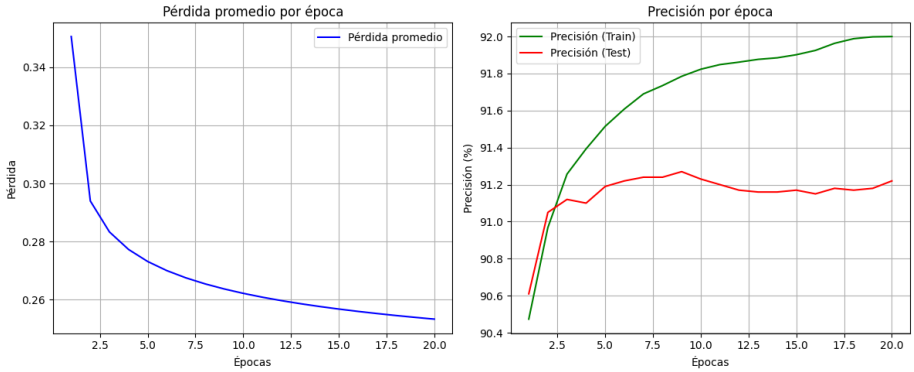

Figura 1: Red Neuronal Simple

Desempeño General

- La precisión en el conjunto de entrenamiento aumenta de forma constante, comenzando en **90.47 %** y alcanzando **92.00 %** después de 20 épocas.
- La precisión en el conjunto de prueba sigue un comportamiento similar, alcanzando **91.22 %** al final.

Tasa de Aprendizaje

- La pérdida promedio disminuye consistentemente durante el entrenamiento, indicando que el modelo está aprendiendo de manera estable.
- Las últimas épocas muestran una disminución más lenta en la pérdida, lo que indica convergencia.

Diferencia entre Entrenamiento y Prueba

La diferencia entre las precisiones de entrenamiento y prueba es mínima, lo que sugiere que el modelo no está sobreajustado.

Limitaciones del Modelo

- La precisión final del 91.22 % en el conjunto de prueba es razonable, pero no sobresaliente, lo que se espera dado que la red es relativamente simple.
- La capacidad del modelo para capturar patrones complejos en los datos es limitada debido a su arquitectura básica.

Resultados de la Red Neuronal Multicapa

se analizan los resultados obtenidos al entrenar una red neuronal multicapa (MLP) para la clasificación de dígitos en el conjunto de datos MNIST. La red consta de múltiples capas ocultas y utiliza funciones de activación `ReLU` y una capa de salida con `softmax`. Se emplearon 100 épocas de entrenamiento con un enfoque de mini-batch.

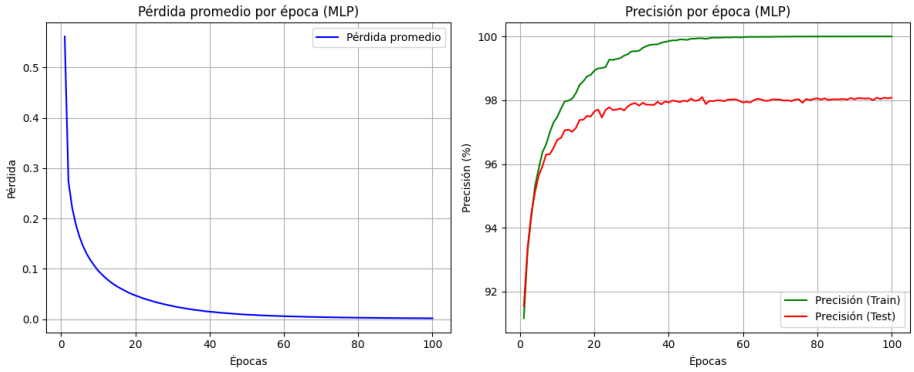

Figura 2: Red Neuronal Multicapa

Análisis de Resultados

La red neuronal multicapa logra un desempeño significativamente mejor que la red neuronal simple:

- La precisión en el conjunto de prueba aumenta hasta un **98.08 %** al final del entrenamiento.
- En el conjunto de entrenamiento, la precisión alcanza un **100 %**, lo que indica que la red ha aprendido completamente los datos de entrenamiento.

Tasa de Aprendizaje y Convergencia

- La pérdida promedio disminuye consistentemente durante el entrenamiento, desde **0.5617** en la primera época hasta **0.0020** en la última.
- Las mejoras en precisión son más pronunciadas en las primeras 20 épocas, con una estabilización hacia las últimas épocas.

Diferencia entre Entrenamiento y Prueba

- La precisión en el conjunto de prueba (98.08 %) es ligeramente inferior a la del conjunto de entrenamiento (100 %).
- Esto sugiere que la red podría estar empezando a sobreajustarse hacia las últimas épocas. Sin embargo, la diferencia es mínima, lo que indica un buen balance entre sesgo y varianza.

Eficiencia del Modelo

La red multicapa, con sus múltiples capas ocultas, logra capturar patrones más complejos que la red simple, lo que se refleja en la mayor precisión en el conjunto de prueba.

El modelo de red neuronal multicapa logra una precisión del **98.08 %** en el conjunto de prueba, lo que representa una mejora significativa en comparación con la red neuronal simple (**91.22 %**). Esto demuestra la eficacia de redes más profundas para tareas de clasificación.

Resultados de la Red Convolucional (CNN)

Este análisis detalla los resultados obtenidos al entrenar una red neuronal convolucional para la clasificación de dígitos en el conjunto de datos MNIST.

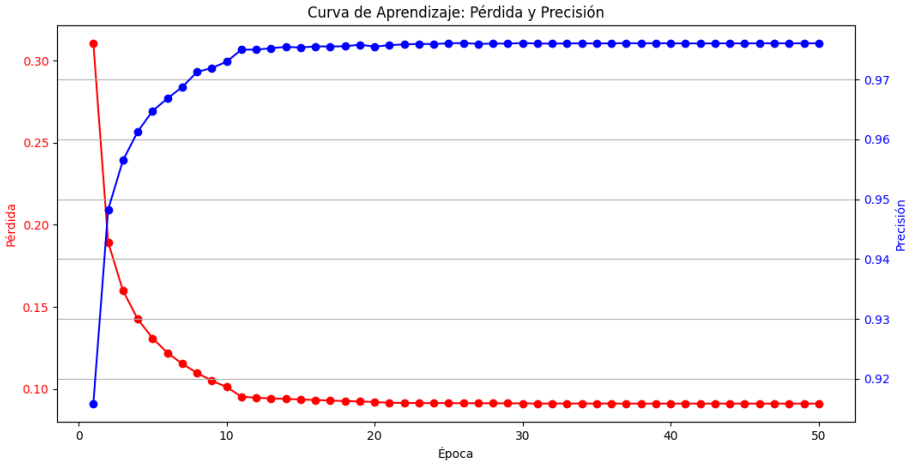

Figura 3: Red Neuronal Convolucional

Análisis del Entrenamiento

- **Inicio del Entrenamiento:** En la primera época, la pérdida promedio fue de 0.3106 y la precisión alcanzó el 91.59 %, indicando un buen rendimiento inicial gracias a la inicialización adecuada de los pesos y filtros convolucionales.
- **Mejoras Rápidas en las Primeras Épocas:** Durante las primeras 10 épocas, se observa una rápida disminución de la pérdida (de 0.3106 a 0.1011), mientras que la precisión aumentó a 97.29 %. Esto sugiere que la CNN aprendió características visuales importantes, como bordes y patrones.
- **Estabilización en las Últimas Épocas:** A partir de la época 20, la pérdida promedio se estabilizó en 0.0910, y la precisión alcanzó el 97.60 %. Esto indica que el modelo llegó a un punto de saturación en su aprendizaje, sin mejoras significativas posteriores.

Resultados en el Conjunto de Prueba

La precisión en el conjunto de prueba fue del **97.28 %**, en línea con la precisión obtenida en el conjunto de entrenamiento (97.60 %). Esto indica:

- **Buen ajuste:** La red no muestra señales de sobreajuste, ya que el rendimiento en el conjunto de prueba es muy similar al del conjunto de entrenamiento.

- **Generalización sólida:** Las características aprendidas por la CNN son lo suficientemente robustas para generalizar correctamente en datos no vistos.

Resultados de la Red Deep Learning con 3 Capas Ocultas

Este análisis detalla los resultados obtenidos al entrenar una red neuronal profunda (Deep Learning) con 3 capas ocultas para la clasificación de dígitos en el conjunto de datos MNIST. La arquitectura de la red es la siguiente:

- **Capa de entrada:** 784 neuronas (28x28 píxeles aplanados).
- **Primera capa oculta:** 512 neuronas con activación `ReLU`.
- **Segunda capa oculta:** 256 neuronas con activación `ReLU`.
- **Tercera capa oculta:** 128 neuronas con activación `ReLU`.
- **Capa de salida:** 10 neuronas (una por clase) con activación `softmax`.

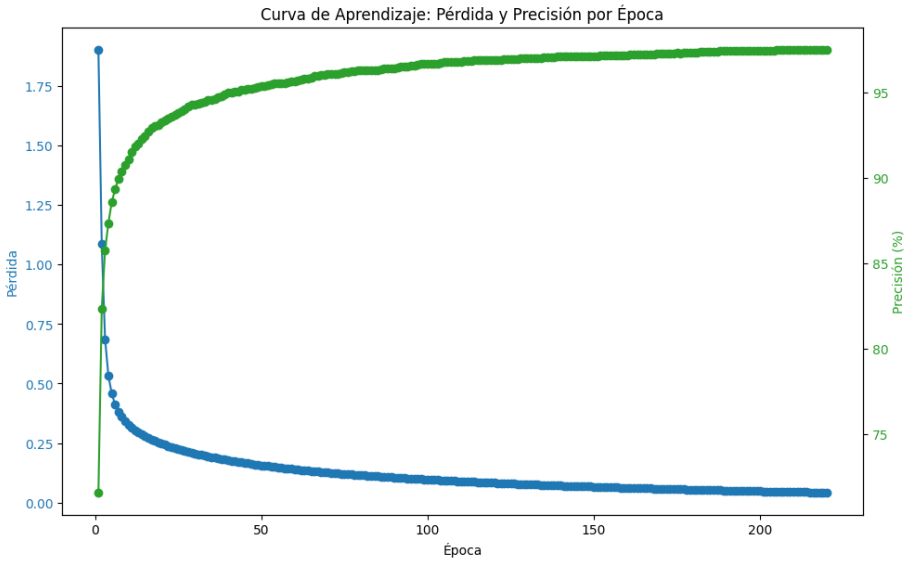

Figura 4: Deep Learning - [254, 254, 254]

Análisis de Resultados

- - La red neuronal alcanzó una precisión del **97.49 %** en el conjunto de prueba al final del entrenamiento, mostrando un excelente desempeño.
- La pérdida promedio en el conjunto de entrenamiento disminuyó de **0.3271** en la época 10 a **0.0421** en la época 220, indicando una convergencia estable y efectiva.
- La pérdida en el conjunto de prueba también mostró una mejora significativa, reduciéndose de **0.3075** a **0.0801**.

Eficiencia del Modelo

- La arquitectura más compacta, en comparación con redes más profundas, mantuvo un alto desempeño, lo que demuestra que la combinación de un diseño bien ajustado y regularización es efectiva.
- El uso de capas ocultas con tamaños decrecientes (`512 → 256 → 128`) facilita la extracción de características relevantes mientras se mitiga el riesgo de sobreajuste.

Comparación con Otras Arquitecturas

- **Red Neuronal Simple:** Esta red supera ampliamente el desempeño de la red simple, la cual alcanzó un máximo de precisión del **91.22 %**.
- **Red con Capas Uniformes:** Aunque esta red tiene un desempeño similar al de la red con capas uniformes (**97.54 %**), su arquitectura más compacta puede ser computacionalmente más eficiente.
- **Red con 4 Capas Ocultas:** La red con 4 capas ocultas alcanzó una precisión del **97.67 %**, siendo ligeramente superior, pero a costa de una mayor complejidad.

Resultados de la Red Deep Learning con 4 Capas Ocultas

En este análisis se presentan los resultados obtenidos al entrenar una red neuronal profunda (Deep Learning) con 4 capas ocultas para la clasificación de dígitos en el conjunto de datos MNIST.

La arquitectura de la red es la siguiente:

- **Capa de entrada:** 784 neuronas (28x28 píxeles aplanados).
- **Primera capa oculta:** 512 neuronas con activación `ReLU`.

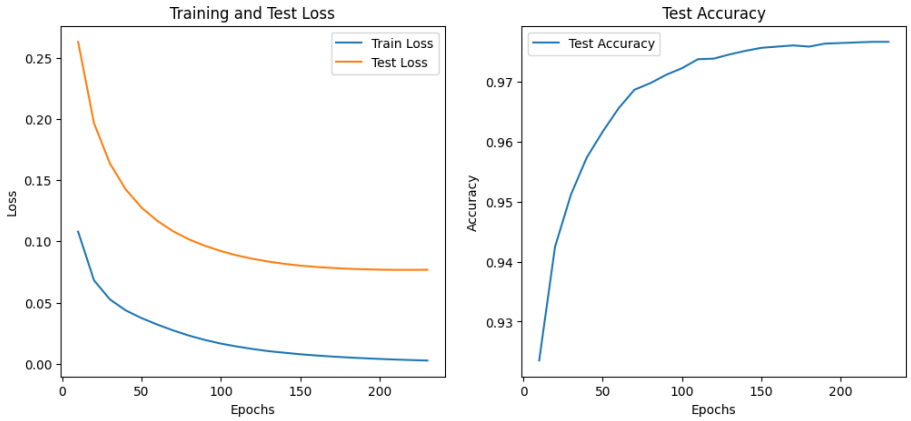

Figura 5: Enter Caption

- **Segunda capa oculta:** 256 neuronas con activación `ReLU`.
- **Tercera capa oculta:** 128 neuronas con activación `ReLU`.
- **Cuarta capa oculta:** 64 neuronas con activación `ReLU`.
- **Capa de salida:** 10 neuronas (una por clase) con activación `softmax`.

El modelo fue entrenado utilizando 230 épocas con un enfoque de mini-batch, optimización basada en descenso de gradiente y regularización para evitar sobreajuste.

Análisis de Resultados

- La precisión en el conjunto de prueba alcanzó un máximo de **97.67 %** en la época 230, mostrando un excelente desempeño.
- La pérdida en el conjunto de entrenamiento disminuyó constantemente desde **0.1078** en la época 10 hasta **0.0026** en la última época.
- La pérdida en el conjunto de prueba también muestra una mejora significativa, reduciéndose de **0.2630** a **0.0768**.

Eficiencia del Modelo

- La arquitectura profunda permite capturar patrones complejos, lo que se refleja en la alta precisión en el conjunto de prueba.
- La reducción consistente en la pérdida indica que el modelo converge adecuadamente.

Comparación con Otras Redes

- **Red Neuronal Simple:** Alcanzó una precisión del 91.22 %, mostrando que su capacidad de aprendizaje es limitada en comparación con redes más profundas.
- **Red Multicapa:** Logró una precisión de 98.08 %, siendo ligeramente superior a esta red. Esto podría deberse a diferencias en regularización o hiperparámetros.

Resultados de la Red Deep Learning con 4 Capas Ocultas de Tamaño Uniforme

En este análisis se presentan los resultados obtenidos al entrenar una red neuronal profunda (Deep Learning) con 4 capas ocultas de tamaño uniforme (256 neuronas cada una) para la clasificación de dígitos en el conjunto de datos MNIST. La arquitectura de la red es la siguiente:

- **Capa de entrada:** 784 neuronas (28x28 píxeles aplanados).
- **Cuatro capas ocultas:** Cada una con 256 neuronas y activación `ReLU`.
- **Capa de salida:** 10 neuronas (una por clase) con activación `softmax`.

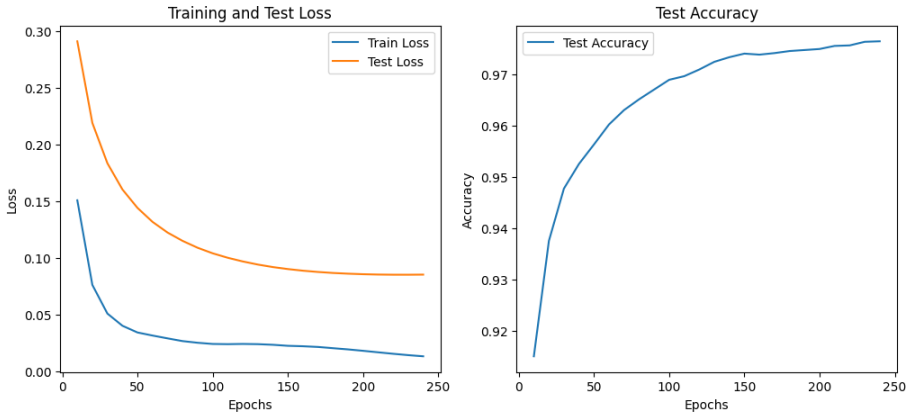

Figura 6: Deep Learning - [128,128,128,128]

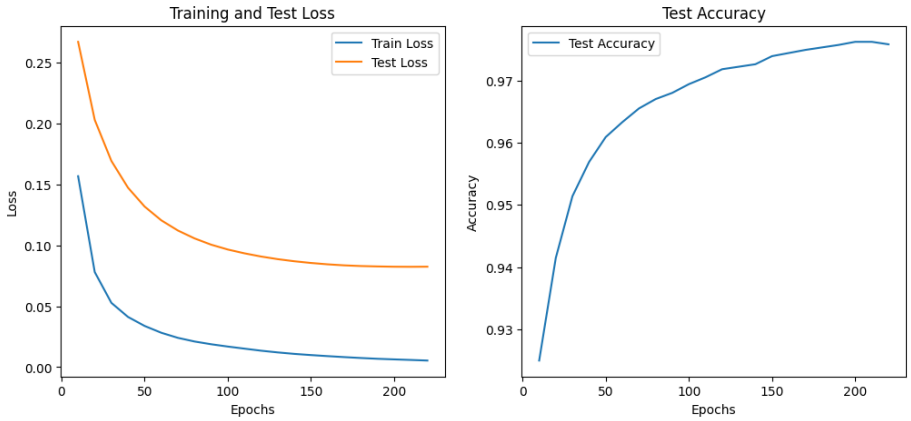

Figura 7: Deep Learning - [254, 254, 254, 254]

Análisis de Resultados

- La red neuronal alcanzó una precisión del **97.54 %** en el conjunto de prueba, lo que representa un excelente desempeño en la tarea de clasificación de dígitos. La pérdida promedio en el conjunto de entrenamiento disminuyó significativamente desde **0.1312** en la época 10 hasta **0.0035** en la última época.
- La pérdida en el conjunto de prueba también mostró una mejora considerable, reduciéndose de **0.2699** a **0.0839**.

Convergencia y Regularización

- La red converge de manera estable a medida que avanzan las épocas, con una mejora continua en la precisión.
- La diferencia entre la pérdida en entrenamiento y en prueba es mínima hacia las últimas épocas, lo que indica un buen balance entre sesgo y varianza.

Comparación con Otras Arquitecturas

- En comparación con la red con arquitectura '[784, 512, 256, 128, 64, 10]', esta red tiene un desempeño ligeramente inferior en términos de precisión final (**97.54 %** frente a **97.67 %**).
- Sin embargo, su arquitectura más simple podría ser más eficiente en términos computacionales y fácil de ajustar.

Resultados de la Red Convolucional con Librerias

Los resultados obtenidos reflejan el desempeño de un modelo convolucional mejorado en la tarea de clasificación de imágenes del conjunto de datos MNIST. Este modelo incorpora técnicas avanzadas como el aumento de datos, regularización, optimización con `AdamW`, y `Batch Normalization`. A continuación, se analizan los resultados clave del entrenamiento y evaluación.

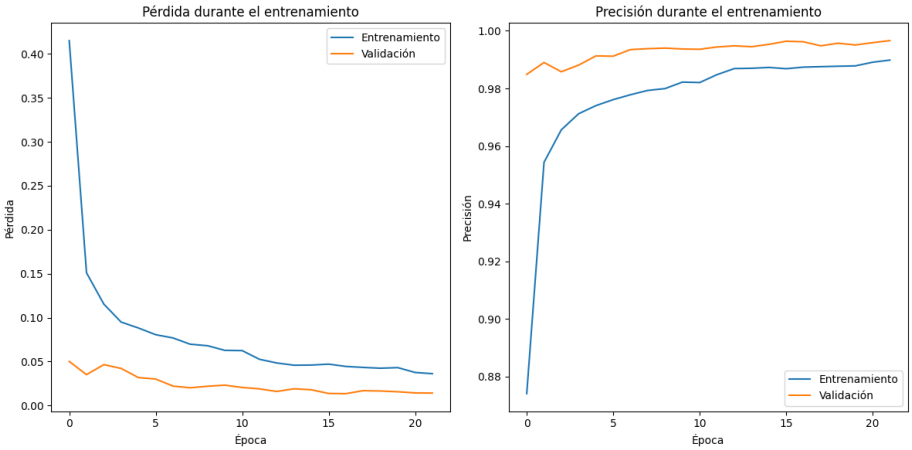

Figura 8: Red Neuronal Convolucional - Con Librerias

Resultados del Entrenamiento

Durante 22 épocas, el modelo mostró un progreso constante tanto en la pérdida (*loss*) como en la precisión (*accuracy*). A continuación, se destacan algunos puntos importantes:

- **Época 1**:
  - Pérdida: **0.8014**.
  - Precisión: **76.19 %**.
  - Precisión de validación: **98.49 %**.
  - Pérdida de validación: **0.0501**.

  Esto muestra una rápida convergencia inicial gracias a un optimizador efectivo y una arquitectura bien diseñada.

- **Época 7**:
  - Pérdida: **0.0758**.

- Precisión: **97.81 %**.
- Precisión de validación: **99.35 %**.
- Pérdida de validación: **0.0221**.

En esta etapa, el modelo ya alcanza un rendimiento competitivo, con un margen mínimo de mejora.

- **Época 16**:
  - Pérdida: **0.0454**.
  - Precisión: **98.77 %**.
  - Precisión de validación: **99.64 %**.
  - Pérdida de validación: **0.0137**.

  Se evidencia una estabilización en las métricas, y el modelo comienza a aproximarse a su rendimiento óptimo.

- **Última época (22)**:
  - Pérdida: **0.0365**.
  - Precisión: **98.97 %**.
  - Precisión de validación: **99.66 %**.
  - Pérdida de validación: **0.0141**.

Evaluación en el Conjunto de Prueba El modelo alcanzó un desempeño sobresaliente en el conjunto de prueba:

- **Pérdida**: **0.0135**.
- **Precisión**: **99.62 %**.

Esto demuestra que el modelo no sólo se adapta bien a los datos de entrenamiento, sino que también generaliza eficazmente a datos no vistos.

Análisis de las Métricas

- La combinación de `Batch Normalization`, `Dropout` y regularización `L2` contribuyó significativamente a la estabilización del entrenamiento y a la reducción del sobreajuste.
- La estrategia de reducción de la tasa de aprendizaje (`ReduceLROnPlateau`) permitió al modelo refinar su rendimiento en las últimas épocas, ajustándose mejor al mínimo global.
- La inclusión de `Data Augmentation` ayudó a ampliar la diversidad del conjunto de datos, mejorando la capacidad del modelo para generalizar.

Resultados de la Red Deep Convolucional v2 con Librerias

En este informe se analizan los resultados obtenidos en un modelo de clasificación de imágenes utilizando el dataset MNIST. El modelo está basado en redes neuronales convolucionales (CNN) y ha sido entrenado y evaluado con técnicas de regularización y optimización adaptativa.

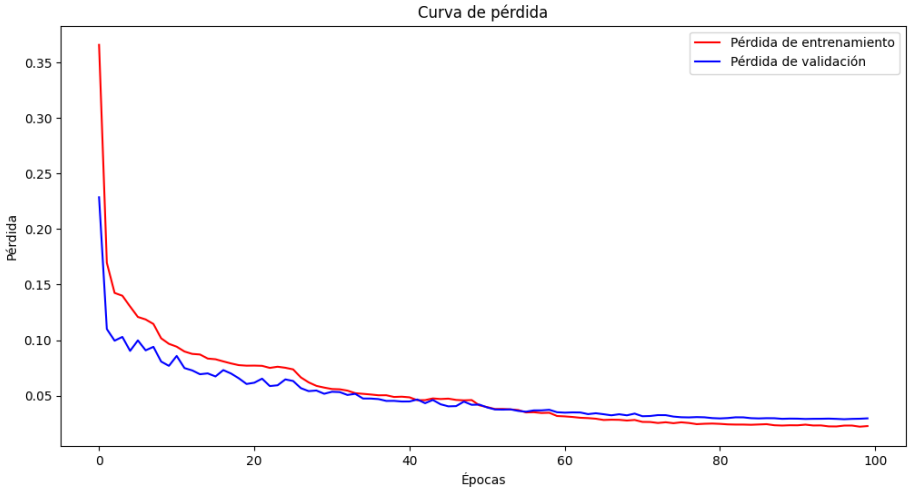

Figura 9: Red Neuronal Convolucional v2 - Loss

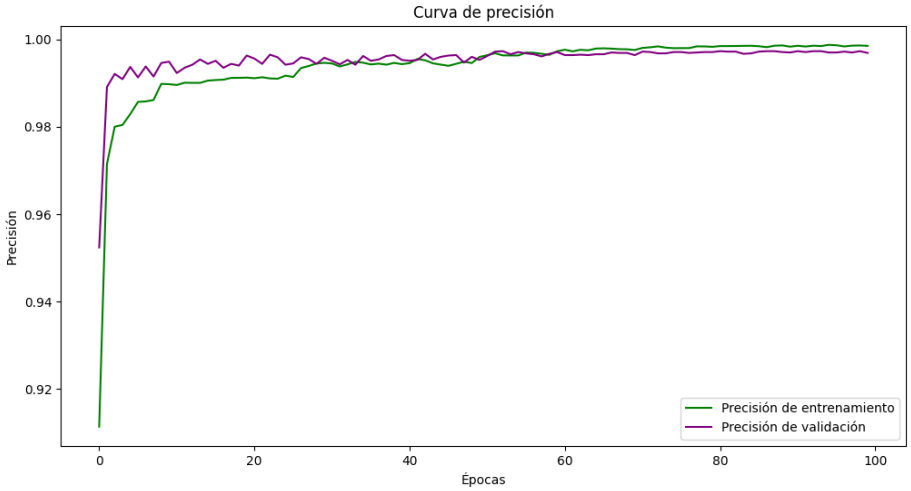

Figura 10: Red Neuronal Convolucional v2 - Precisión

Arquitectura del Modelo

El modelo utilizado consta de tres bloques convolucionales seguidos de capas de 'BatchNormalization', 'Dropout' y una capa densa. Además, se implementó un esquema de optimización con el optimizador AdamW y la tasa de aprendizaje fue ajustada durante el entrenamiento utilizando la técnica 'ReduceLROnPlateau'.

Evolución de la Precisión y Pérdida

A continuación se presenta un análisis de la evolución de las métricas de entrenamiento, validación y prueba a lo largo de las épocas:

- **Épocas 1-10:** La precisión de entrenamiento mejora de **81.21 %** a **99.06 %**, mientras que la precisión de validación aumenta de **95.24 %** a **99.49 %**. Esto muestra que el modelo empieza a aprender los patrones básicos de los datos de entrenamiento.
- **Épocas 11-40:** La precisión de entrenamiento y validación se mantiene en niveles altos, alcanzando **99.64 %** en la validación en la época 40. La pérdida en validación también sigue disminuyendo, indicando que el modelo generaliza bien y no está sobreajustado.
- **Épocas 41-100:** La precisión de validación alcanza su valor máximo de **99.73 %** y la precisión de prueba es **99.72 %**, con una pérdida de **0.0282** en el conjunto de prueba. Este resultado indica que el modelo está altamente optimizado y generaliza bien a nuevos datos.

Métricas de Evaluación Final

La evaluación final del modelo en el conjunto de prueba mostró los siguientes resultados:

- **Precisión de prueba:** **99.72 %**
- **Pérdida de prueba:** **0.0282**
- **Imágenes mal clasificadas:** **28 imágenes**.

Conclusiones

Red Simple

El modelo de red simple mostró una tasa de precisión relativamente alta desde el inicio, alcanzando una precisión de 90.47 % en la primera época y mejorando progresivamente hasta alcanzar el 92.00 % al final del entrenamiento. La pérdida

promedio disminuyó consistentemente, lo que indica que el modelo fue capaz de aprender las características relevantes de las imágenes. Sin embargo, su tasa de mejora se desaceleró hacia las últimas épocas, lo que sugiere que el modelo alcanzó un punto de saturación y no experimentó mejoras significativas más allá de la época 20.

Red Multicapa

La red multicapa mostró una mejora considerable en comparación con la red simple. En las primeras épocas, la precisión aumentó rápidamente, alcanzando 91.17 % en la primera época y mejorando a un ritmo constante hasta llegar a una precisión de 100 % en las últimas épocas durante el entrenamiento. La pérdida promedio disminuyó de manera sustancial, alcanzando un valor de 0.0020 en la época 100, lo que indica que el modelo continuó aprendiendo de manera efectiva durante el entrenamiento. Este modelo, por lo tanto, demostró un alto rendimiento en términos de precisión y minimización de la pérdida.

Red Convolucional

El modelo convolucional mostró un rendimiento impresionante en términos de precisión y reducción de la pérdida. En la primera época, la precisión fue de 91.59 % con una pérdida de 0.3106, lo cual es relativamente bueno para un modelo convolucional. A medida que avanzaba el entrenamiento, se observó una rápida mejora en la precisión, alcanzando un valor de 97.29 % en la época 10, mientras que la pérdida disminuyó rápidamente. Hacia la época 20, la precisión se estabilizó en 97.60 % y la pérdida se mantuvo en 0.0910, lo que sugiere que el modelo había aprendido eficazmente las características visuales y patrones de las imágenes. A diferencia de los otros modelos, el modelo convolucional no mostró un sobreajuste evidente, ya que la precisión de validación se mantuvo alta, lo que subraya su capacidad para generalizar bien.

Red con 4 Capas Ocultas

En las primeras épocas, el modelo muestra una mejora significativa en la precisión. La precisión aumenta desde 92.24 % en la época 10 hasta 97.54 % en la época 220. La pérdida disminuye gradualmente, lo que indica una mejora constante en el rendimiento. La pérdida final es 0.0035, con una pérdida de prueba de 0.0839. La precisión de prueba alcanza un valor máximo de 97.54 % en la última época.

El comportamiento de la precisión de validación es muy estable, sugiriendo que el modelo no está sobreajustando y tiene una capacidad sólida para generalizar.

Deep Learning con 3 Capas Ocultas

La precisión mejora a un ritmo constante, comenzando en 91.11 % en la época 10 y alcanzando 97.49 % en la época 220. La pérdida también disminuye, pero a un

ritmo más lento en comparación con el primer modelo. La pérdida final es 0.0421, con una pérdida de prueba de 0.0801. La precisión de prueba mejora de manera consistente hasta estabilizarse cerca del 97.49 % al final del entrenamiento.

La precisión se estabiliza más lentamente, lo que sugiere que el modelo tarda un poco más en aprender los patrones de los datos en comparación con la primera implementación.

Red Convolucional con Librerias

Rendimiento General: Ambos modelos muestran un rendimiento excelente, alcanzando precisiones de validación cercanas al 99 %, lo que indica que han aprendido eficazmente los patrones de los datos.

Evolución del Modelo: Ambos modelos muestran una mejora constante en términos de precisión de validación y reducción de la pérdida de validación. Sin embargo, la Red Convolucional v2 con librerías muestra un mejor ajuste en la precisión de prueba, alcanzando 99.72 % en comparación con el 98.97 % de la Red Convolucional con librerías. Estabilidad y Generalización: El modelo v2 muestra una mejor generalización en los datos de prueba, sin sobreajuste, mientras que el primer modelo tiene una ligera mayor estabilidad en términos de precisión.

Optimización: Ambos modelos son altamente optimizados y muestran un buen rendimiento tanto en el conjunto de entrenamiento como en el de prueba. Sin embargo, el modelo v2 sigue siendo ligeramente superior debido a su capacidad para mantener un nivel más alto de precisión de validación y prueba a lo largo de más épocas.