# TRABAJO PRÁCTICO NRO 3

APRENDIZAJE AUTOMÁTICO 2

FECHA DE ENTREGA: 10 DE OCTUBRE DE 2024

PROFESOR: ING. JORGE CEFERINO VALDEZ

Autor: Fernanda Cader

Las redes neuronales convolucionales (CNN) son herramientas fundamentales en el campo del aprendizaje profundo, especialmente en la clasificación de imágenes y otras aplicaciones de visión por computadora. Estas redes se implementan fácilmente utilizando TensorFlow y Keras, dos de las bibliotecas más populares para el desarrollo de modelos de aprendizaje automático.

**Estructura de una CNN**

Una CNN típicamente consta de varias capas que trabajan en conjunto para procesar imágenes:

1. **Capas convolucionales**: Utilizan filtros (o kernels) que se deslizan sobre la imagen para detectar características locales, como bordes y texturas. Esta operación ayuda a reducir la dimensionalidad de la imagen sin perder información relevante.
2. **Activación**: Se suele utilizar la función de activación ReLU (Rectified Linear Unit) para introducir no linealidades en el modelo, permitiendo a la red aprender patrones más complejos.
3. **Agrupamiento (Pooling)**: Después de las capas de convolución, se aplica el pooling (por ejemplo, MaxPooling) para reducir aún más el tamaño de las representaciones y disminuir la carga computacional. Esto también ayuda a hacer el modelo más robusto frente a pequeñas variaciones en la imagen.
4. **Capas densas (Fully Connected)**: En la parte final de la red, se utilizan capas totalmente conectadas para realizar la clasificación basada en las características extraídas por las capas anteriores. Aquí se combinan las activaciones para predecir la clase de la imagen.

**Proceso de entrenamiento y evaluación**

Para construir y entrenar una CNN en TensorFlow y Keras, se siguen varios pasos fundamentales:

- **Definir el modelo**: Se construye la arquitectura de la red utilizando las capas descritas anteriormente.
- **Compilación**: Esto incluye la selección de un optimizador, una función de pérdida y métricas para evaluar el rendimiento del modelo. Por ejemplo, se puede usar Adam como optimizador y la entropía cruzada como función de pérdida para tareas de clasificación.
- **Entrenamiento**: Se ajustan los pesos de la red utilizando un conjunto de datos de entrenamiento. Durante este proceso, se puede aplicar técnicas de regularización para prevenir el sobreajuste.
- **Evaluación**: Finalmente, se evalúa el modelo con un conjunto de prueba para medir su precisión y capacidad de generalización.

**Aplicaciones prácticas**

Las CNN son ampliamente utilizadas en diversas aplicaciones de la vida real:

- **Reconocimiento de imágenes**: Permiten clasificar y etiquetar automáticamente imágenes, lo que es útil en redes sociales y aplicaciones de fotografía.
- **Detección de objetos**: Se utilizan en sistemas de seguridad y vigilancia, así como en vehículos autónomos para identificar y localizar objetos en su entorno.
- **Segmentación de imágenes**: Permiten identificar regiones específicas dentro de una imagen, útil en aplicaciones médicas y de análisis de imágenes.

En resumen, las redes neuronales convolucionales, junto con herramientas como TensorFlow y Keras, han revolucionado la manera en que abordamos problemas de visión por computadora, facilitando la creación de modelos precisos y eficientes para la clasificación y el análisis de imágenes.

## Informe sobre el Clasificador de Dígitos Manuscritos Usando CNN con el Dataset MNIST

### 1. Introducción

La clasificación de imágenes es una de las tareas más comunes en el campo del aprendizaje automático y la visión por computadora. Este informe se centra en el problema de reconocer dígitos manuscritos, un desafío clásico en la comunidad de aprendizaje automático. Para ello, se utilizó el dataset MNIST, que contiene 70,000 imágenes de dígitos del 0 al 9. El objetivo principal es construir un modelo que pueda clasificar correctamente cada imagen en su respectiva categoría de dígito.

### 2. Carga y Preprocesamiento de Datos

### 2.1 Carga de Datos

Se utilizó la función mnist.load_data() de Keras para cargar el dataset MNIST, que ya está dividido en conjuntos de entrenamiento y prueba. Esto permite evaluar el modelo en un conjunto de datos que no ha sido visto durante el entrenamiento.

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

### 2.2 Preprocesamiento de Datos

El preprocesamiento es crucial para asegurar que el modelo funcione correctamente. Las etapas de preprocesamiento incluyeron:

- **Redimensionamiento de Imágenes**: Las imágenes originales de 28x28 píxeles se redimensionaron para incluir un canal adicional (grayscale), resultando en dimensiones de (28, 28, 1). Esto se logra mediante la siguiente línea de código:

train_images = train_images.reshape((train_images.shape[0], 28, 28, 1)).astype('float32')

test_images = test_images.reshape((test_images.shape[0], 28, 28, 1)).astype('float32')

**Normalización**: Los valores de los píxeles, que oscilan entre 0 y 255, se normalizaron a un rango de [0, 1] dividiendo por 255.0. Esto ayuda a acelerar la convergencia durante el entrenamiento y se realizó de la siguiente manera:

train_images /= 255.0

test_images /= 255.0

**Conversión de Etiquetas**: Las etiquetas de los dígitos se convirtieron en un formato de codificación one-hot, facilitando así la clasificación multinomial. Este paso se ejecuta con:

train_labels = to_categorical(train_labels, 10)

test_labels = to_categorical(test_labels, 10)

### 3. Diseño del Modelo

El modelo se construyó utilizando la API secuencial de Keras, lo que permite apilar capas de forma lineal. El diseño del modelo incluyó varias decisiones clave:

#### 3.1 Estructura de la Red

- **Capas Convolucionales**: Se implementaron tres capas convolucionales. Estas capas son esenciales para extraer características jerárquicas de las imágenes:
    - **Primera Capa**: 32 filtros de tamaño 3x3 con función de activación ReLU. Esta capa comienza a captar características básicas de los dígitos, como bordes y líneas.
    - **Segunda Capa**: 64 filtros de tamaño 3x3 con activación ReLU. Al agregar más filtros, se espera que la red reconozca patrones más complejos.
    - **Tercera Capa**: También 64 filtros de tamaño 3x3. Esta capa finaliza la fase de extracción de características.

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

- **Capas de MaxPooling**: Se aplicaron después de las capas convolucionales para reducir la dimensionalidad y extraer las características más relevantes. MaxPooling ayuda a reducir el tamaño espacial de las características, lo que a su vez disminuye la carga computacional y el riesgo de sobreajuste.

#### 3.2 Capa de Aplanamiento y Capas Densas

- **Aplanamiento**: La salida tridimensional de la última capa convolucional se convierte en un vector unidimensional con la capa Flatten().
- **Capas Densas**:

- Una capa densa con 64 neuronas y activación ReLU se agregó para permitir que la red aprenda combinaciones complejas de características extraídas.
- La capa de salida tiene 10 neuronas (una por cada dígito del 0 al 9) y utiliza activación softmax, que proporciona una probabilidad para cada clase.

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(10, activation='softmax'))

## 4. Compilación y Entrenamiento del Modelo

### 4.1 Compilación

El modelo se compiló con el optimizador Adam, que es popular por su capacidad para manejar problemas de optimización no estacionarios, y se utilizó la función de pérdida categorical_crossentropy para problemas de clasificación múltiple. Las métricas utilizadas fueron la precisión.

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

### 4.2 Entrenamiento

El modelo se entrenó durante 5 épocas, utilizando un tamaño de lote de 64 y separando el 20% de los datos de entrenamiento para la validación. Este enfoque permite verificar el rendimiento del modelo en datos no vistos durante el entrenamiento. **Una época en el entrenamiento de una red neuronal es una pasada completa del conjunto de datos de entrenamiento a través de la red.** Es decir, se presentan todas las muestras de entrenamiento a la red una sola vez en una época.

history = model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_split=0.2)

## 5. Resultados del Entrenamiento

Los resultados del entrenamiento se monitorearon y visualizaron a través de gráficos de precisión y pérdida.

- **Precisión**: La precisión en los datos de prueba fue de **0.9901**, lo que indica que el modelo clasifica correctamente el 99.01% de los dígitos en el conjunto de prueba.
- **Pérdida**: La pérdida final fue baja, con valores que sugieren que el modelo se ajustó bien a los datos.

### 5.1 Visualización de Resultados

- **Precisión durante el Entrenamiento**: El gráfico de precisión muestra una mejora constante en la precisión tanto en los conjuntos de entrenamiento como de validación.

- **Pérdida durante el Entrenamiento**: El gráfico de pérdida indica una disminución significativa de la pérdida, lo que sugiere que el modelo está aprendiendo de manera efectiva.

### 6. Evaluación del Modelo

Se evaluó el modelo utilizando el conjunto de prueba y se calcularon métricas adicionales para entender su rendimiento.

### 6.1 Informe de Clasificación

Se generó un informe de clasificación que incluye precisión, recall y F1-score para cada clase (dígito). Estas métricas son cruciales para comprender no solo cuántas predicciones son correctas, sino también cuántas fueron erróneas en términos de clasificaciones positivas y negativas.

print(classification_report(true_labels, predicted_labels))

Los resultados mostraron que el modelo tiene una precisión y recall muy altos para todas las clases, con un F1-score promedio de alrededor de **0.99**.

### 6.2 Matriz de Confusión

Se calculó la matriz de confusión, que proporciona una visualización clara de cómo el modelo confunde diferentes clases. Esta matriz reveló que el modelo comete pocos errores, lo que indica un buen rendimiento general.

conf_matrix = confusion_matrix(true_labels, predicted_labels)

### 7. Conclusiones y Futuras Direcciones

El modelo CNN desarrollado logró un rendimiento sobresaliente en la clasificación de dígitos manuscritos utilizando el dataset MNIST. La alta precisión y las bajas tasas de error en la matriz de confusión demuestran la efectividad de las redes neuronales convolucionales para problemas de reconocimiento de imágenes.

Sin embargo, hay varias direcciones futuras que se pueden explorar:

- **Aumento de Datos**: Implementar técnicas de aumento de datos para mejorar la generalización del modelo.
- **Ajuste de Hiperparámetros**: Probar diferentes arquitecturas de red y ajustar hiperparámetros para optimizar aún más el rendimiento.
- **Transfer Learning**: Utilizar modelos preentrenados en conjuntos de datos más grandes y transferir su conocimiento al problema de clasificación de dígitos.

Este proyecto no solo proporciona una base sólida para la clasificación de imágenes, sino que también establece un marco para abordar problemas más complejos en el campo del aprendizaje automático y la visión por computadora.