



SESIÓN REDES NEURONALES CONVOLUTIVAS

CONTENIDOS:

- Qué es una red neuronal convolutiva.
 - Qué es una convolución.
 - Campo de utilización de las redes neuronales convolutivas.
- Etapas en las RNC:
 - Operación de convolución y capa relu.
 - Capa de pooling.
 - Flattening.
 - Full connection.
- Implementación de redes neuronales convolutivas en Python.
- Evaluación y optimización de la red neuronal convolutiva.

QUÉ ES UNA RED NEURONAL CONVOLUTIVA

Una **red neuronal convolutiva (CNN, por sus siglas en inglés)** es un tipo de red neuronal diseñada específicamente para procesar datos con una estructura de cuadrícula, como imágenes. Su arquitectura permite detectar patrones espaciales y características jerárquicas en los datos, lo que las hace especialmente útiles en **procesamiento de imágenes, visión por computadora y reconocimiento de patrones**.

Las CNN utilizan **capas convolucionales** para extraer características relevantes de una imagen y reducir la cantidad de parámetros, mejorando la eficiencia y precisión en tareas de clasificación, segmentación y detección de objetos.

Qué es una convolución

La **convolución** es una operación matemática que se aplica a una matriz de datos (como una imagen) utilizando un filtro o kernel. Su propósito es extraer características específicas, como bordes, texturas y formas.

Ejemplo:

- Si tenemos una imagen en escala de grises representada por una matriz de píxeles, un **filtro de detección de bordes** puede recorrer la imagen y destacar las áreas donde hay cambios bruscos en la intensidad de los píxeles.
- Este proceso se realiza mediante una **operación de producto punto** entre el filtro y una región de la imagen, desplazándose por toda la imagen en pequeños pasos llamados *strides*.

Beneficios de la convolución:

- ✓ Reduce la cantidad de parámetros en comparación con redes totalmente conectadas.
- ✓ Extrae características relevantes de forma automática.
- ✓ Mantiene la información espacial en los datos.

Campo de utilización de las redes neuronales convolutivas

Las CNN se utilizan en diversas áreas, principalmente en **visión por computadora**, pero también han sido adaptadas a otros dominios.

Visión por computadora

- Reconocimiento facial (Ej: Face ID en iPhones)
- Detección y clasificación de objetos (Ej: Autos autónomos)
- Segmentación de imágenes médicas (Ej: Diagnóstico de enfermedades en radiografías)

Procesamiento de imágenes y video

- Mejora de imágenes y restauración (Ej: Eliminación de ruido en fotografías)
- Análisis de video en tiempo real (Ej: Seguridad y vigilancia con cámaras inteligentes)

Procesamiento de texto y lenguaje natural

- OCR (Reconocimiento Óptico de Caracteres)
- Extracción de información de documentos

Juegos y simulaciones

- Generación de imágenes realistas con redes generativas
- Interacción en entornos 3D

Otras aplicaciones

- Reconocimiento de defectos en manufactura industrial
- Predicción en datos meteorológicos y astronómicos

ETAPAS EN LAS REDES NEURONALES CONVOLUTIVAS (CNN o RNC)

Las **Redes Neuronales Convolutivas (RNC)** procesan imágenes en varias etapas para extraer características y tomar decisiones. A continuación, se describen sus principales etapas:

Operación de convolución

La convolución es la **primera** etapa en una CNN. Se aplica un **filtro (o kernel)** sobre la imagen para extraer características como bordes, texturas y formas.

Proceso:

1. Se desplaza un filtro pequeño (matriz de pesos) sobre la imagen.
2. Se calcula el **producto punto** entre el filtro y la región de la imagen.
3. Se genera un **mapa de características (feature map)** con la información más relevante.

Ejemplo de una convolución con un filtro 3x3 sobre una imagen en escala de grises:

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

1	0	-1
1	0	-1
1	0	-1

Ilustración 1 Convolución con un filtro 3x3



El filtro recorrerá la imagen, aplicando operaciones de multiplicación y sumando los valores resultantes.

Capa ReLU (Rectified Linear Unit)

Después de la convolución, la capa **ReLU** introduce **no linealidad** en la red, eliminando valores negativos y acelerando el aprendizaje.

Función matemática:

$$f(x) = \max(0, x)$$

Ilustración 2 Función matemática ReLu

Beneficios de ReLu:

- ✓ Elimina valores negativos que no aportan información.
- ✓ Reduce el problema de gradientes desaparecidos.
- ✓ Agiliza el entrenamiento de la red.

Capa de pooling (Submuestreo o Downsampling)

Después de la convolución y ReLU, la red aplica **pooling** para reducir la dimensionalidad del mapa de características.

Tipos de Pooling:

- ◊ **Max Pooling** (el más común) → Se toma el valor máximo de una región de la imagen.
- ◊ **Average Pooling** → Se calcula el promedio de una región.

Ejemplo de **Max Pooling 2x2 con stride 2**:

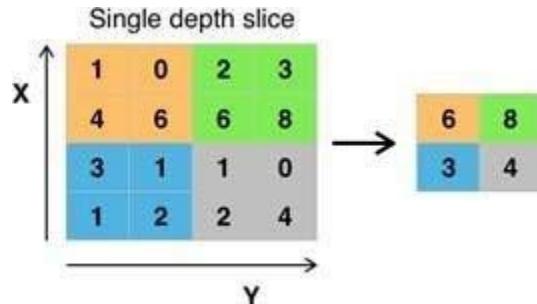


Ilustración 3 Ejemplo Max Pooling 2 x 2

Beneficios del Pooling:

- ✓ Reduce el número de parámetros, evitando sobreajuste.
- ✓ Mantiene la información relevante.
- ✓ Mejora la eficiencia computacional.

Flattening (Aplanado)

Después de las capas convolucionales y de pooling, los datos aún tienen una estructura matricial.

Flattening convierte esta matriz en un **vector unidimensional** para poder alimentar capas completamente conectadas.

Propósito de Flattening:

- ✓ Permite conectar la CNN con una capa densa.
- ✓ Convierte la información en un formato adecuado para clasificación.

Full connection

Después de convertir los datos en un vector, se pasa a una **capa densa (fully connected layer)**, similar a una red neuronal tradicional.

Funcionamiento:

1. Cada neurona recibe conexiones de todas las neuronas anteriores.
2. Se aplican pesos y sesgos para hacer predicciones.
3. Se usa una función de activación como **softmax** para clasificación.

Beneficios de la capa fully connected:

- ✓ Permite tomar decisiones finales basadas en las características extraídas.
- ✓ Relaciona la información extraída en las capas anteriores.

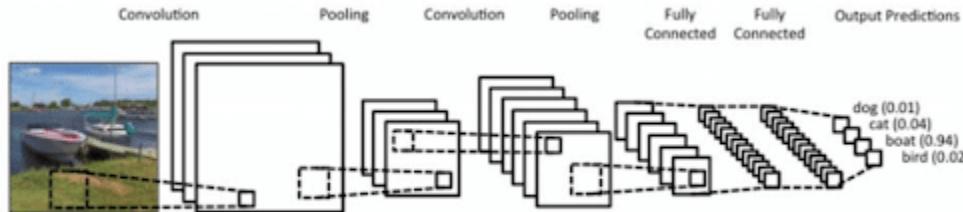


Ilustración 4 Etapas Redes Neuronales Convolutivas

IMPLEMENTACIÓN DE REDES NEURONALES CONVOLUTIVAS EN PYTHON

Para este ejemplo, entrenaremos una CNN para clasificar imágenes del conjunto de datos **CIFAR-10**, que contiene 60,000 imágenes en 10 categorías (aviones, automóviles, pájaros, gatos, etc.).

- ✓ Procura tener instaladas las dependencias de TensorFlow y Keras.

Paso 1: Importación de librerías

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
```

Paso 2: Cargar y preprocesar el dataset

```
# Cargar el dataset CIFAR-10
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Normalizar los valores de los píxeles a un rango de 0 a 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# Convertir las etiquetas a one-hot encoding
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

Paso 3: Definir la arquitectura de la CNN

```
model = keras.Sequential([
    # Primera capa convolucional
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2,2)),

    # Segunda capa convolucional
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),

    # Aplanado y Fully Connected
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax') # 10 clases
])
```

Paso 4: Compilar y entrenar el modelo

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

# Entrenar la CNN
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test))
```

Paso 5: Evaluar el modelo

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'\nPrecisión en test: {test_acc:.4f}')
```

Paso 6: Visualizar los resultados

```
# Graficar la pérdida y precisión
plt.plot(history.history['accuracy'], label='Precisión en entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión en validación')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()
plt.show()
```

EVALUACIÓN Y OPTIMIZACIÓN DE LA RED NEURONAL CONVOLUTIVA

Una vez que el modelo está entrenado, es importante evaluar su rendimiento y mejorarlo si es necesario.

Evaluación del rendimiento

Para evaluar la CNN, se pueden usar métricas como:

- ✓ **Precisión (accuracy)**
- ✓ **Matriz de confusión**
- ✓ **Curva de pérdida y precisión**



Ejemplo de matriz de confusión:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Predicciones
y_pred = np.argmax(model.predict(x_test), axis=1)
y_true = np.argmax(y_test, axis=1)

# Matriz de confusión
cm = confusion_matrix(y_true, y_pred)

# Graficar la matriz de confusión
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10), yticklabels=range(10))
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.show()
```

Optimización del modelo

Si el modelo no tiene un buen rendimiento, se pueden aplicar estas estrategias:

- ◊ **Aumento de datos (Data Augmentation)**

Para mejorar la generalización, se pueden crear imágenes artificialmente transformadas:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)

datagen.fit(x_train)
```

Luego, entrenar el modelo con los datos aumentados:

```
model.fit(datagen.flow(x_train, y_train, batch_size=64), epochs=10, validation_data=(x_test, y_test))
```

- ◊ **Regularización con Dropout**

Evita el sobreajuste apagando aleatoriamente neuronas durante el entrenamiento.

layers.Dropout(0.5)

◊ Cambio de Optimizador

Si el modelo no converge bien, probar otro optimizador como SGD o RMSprop:

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

◊ Ajuste de Hiperparámetros

Experimentar con:

- Número de **capas y neuronas**
- **Tamaño de los filtros** en la convolución
- **Tasa de aprendizaje**

Se pueden optimizar con **Grid Search** o **Optuna**:

```
import optuna

def objective(trial):
    filters = trial.suggest_int("filters", 16, 128)
    lr = trial.suggest_loguniform("lr", 1e-4, 1e-2)

    model = keras.Sequential([
        layers.Conv2D(filters, (3,3), activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer=keras.optimizers.Adam(lr=lr),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    history = model.fit(x_train, y_train, epochs=5, batch_size=64, verbose=0, validation_data=(x_test, y_test))

    return max(history.history['val_accuracy'])

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10)
print("Mejores hiperparámetros:", study.best_params)
```