Arquitectura de las Redes neuronales convolucionales

https://poloclub.github.io/cnn-explainer/#:~:text=Padding%20is%20often%20necessary%20when,build%20depper%2C%20higher%20performing%20networks.

https://setosa.io/ev/image-kernels/

La arquitectura de una red convolucional (Convolutional Neural Network, CNN) es utilizada comúnmente en problemas de visión por computadora. Veamos las características de esta red neuronal en detalle:

Código

(Importaciones en anexo, abajo)

```
network = Sequential()
network.add(Conv2D(32, (3,3), input_shape = (64,64,3), activation='relu'))
network.add(MaxPooling2D(pool_size=(2,2)))

network.add(Conv2D(32, (3,3), activation='relu'))
network.add(MaxPooling2D(pool_size=(2,2)))

network.add(Flatten())

network.add(Dense(units = 7317, activation='relu'))
network.add(Dense(units = 7317, activation='relu'))
network.add(Dense(units = 3, activation='softmax'))
```

Explicación del código

1. Estructura de la red:
    - La red se crea utilizando el modelo secuencial (Sequential()), lo que significa que las capas se apilan secuencialmente una encima de la otra.
    - La red consiste en varias capas convolucionales, seguidas de capas de pooling y capas completamente conectadas (densas).
2. Capas convolucionales:
    - La primera capa convolucional (Conv2D) tiene 32 filtros de tamaño 3x3 y utiliza la función de activación ReLU (activation='relu'). Recibe una entrada con forma (64, 64, 3), lo que indica que espera imágenes de entrada de 64x64 píxeles con 3 canales de color (RGB).

- Después de la primera capa convolucional, se aplica una capa de pooling (MaxPooling2D) con un tamaño de ventana de 2x2 para reducir el tamaño de la representación y extraer características importantes.

3. Capas convolucionales adicionales:
   - Se añade otra capa convolucional similar a la primera, también con 32 filtros de tamaño 3x3 y función de activación ReLU.
   - Luego se aplica otra capa de pooling para reducir el tamaño de la representación nuevamente.
4. Capa de aplanamiento:
   - Después de las capas convolucionales y de pooling, se agrega una capa de aplanamiento (Flatten()) para convertir los mapas de características 2D en un vector 1D. Esto permite que las características extraídas se alimenten a capas densamente conectadas.
5. Capas densas (completamente conectadas):
   - Se añaden tres capas densas (Dense) después de la capa de aplanamiento. Cada capa tiene una función de activación ReLU (activation='relu').
   - Las dos primeras capas densas tienen una gran cantidad de unidades (7317) y la última capa densa tiene 3 unidades, correspondientes a las clases de salida del problema. La función de activación de la última capa es softmax, lo que indica que se utiliza para la clasificación multiclase.

Capas de Pooling

Las capas de pooling son capas que se utilizan en las redes neuronales convolucionales para reducir la dimensionalidad espacial de las representaciones de las características obtenidas por las capas convolucionales. Su objetivo principal es disminuir la cantidad de parámetros y operaciones en la red, al mismo tiempo que conservan las características más relevantes de la imagen.

La operación de pooling se realiza deslizando una ventana (también conocida como kernel o filtro) sobre la entrada y tomando una estadística resumida de los valores de la ventana. La estadística más comúnmente utilizada es el valor máximo (Max Pooling), donde se selecciona el valor máximo dentro de la ventana. Sin embargo, también existen otros tipos de pooling, como el promedio (Average Pooling), donde se calcula el promedio de los valores dentro de la ventana.

Al aplicar el pooling, se logran dos efectos principales:

1. Reducción de dimensionalidad: El pooling reduce la resolución espacial de la representación de las características, ya que reduce el tamaño de las ventanas de muestreo. Esto reduce la cantidad de parámetros en la red y acelera el procesamiento.
2. Invarianza a pequeñas traslaciones: El pooling busca extraer las características más importantes independientemente de su ubicación exacta en la imagen. Al seleccionar el valor máximo o promedio dentro de una ventana, el pooling permite que las características se mantengan incluso si la ubicación precisa de la característica cambia ligeramente en la imagen.

Optimizadores:

Existen varios optimizadores disponibles en TensorFlow y Keras para ajustar los pesos de un modelo durante el entrenamiento.

Estos son solo algunos ejemplos de optimizadores disponibles en TensorFlow y Keras. Cada optimizador tiene sus propias características y es más adecuado para diferentes tipos de problemas y conjuntos de datos. La elección del optimizador depende del caso de uso específico y puede requerir ajustes y experimentación para obtener los mejores resultados.

1. Gradiente descendente estocástico (SGD): Es el optimizador más básico y ampliamente utilizado. Actualiza los pesos utilizando el gradiente negativo de la función de pérdida multiplicado por una tasa de aprendizaje constante.
2. Adam: Es un optimizador muy popular que combina características del algoritmo de gradiente descendente estocástico y del algoritmo Adagrad. Ajusta automáticamente la tasa de aprendizaje y mantiene una adaptación individualizada para cada parámetro.
3. RMSprop: Es un optimizador que adapta la tasa de aprendizaje para cada parámetro según la magnitud de las actualizaciones anteriores. Es útil en situaciones en las que la tasa de aprendizaje global puede ser demasiado grande y causar divergencia.
4. Adagrad: Es un optimizador que adapta la tasa de aprendizaje para cada parámetro en función de su historial de gradientes. Es útil en problemas en los que los parámetros raros deben recibir actualizaciones más frecuentes.
5. Adadelta: Es un optimizador que se basa en el algoritmo Adagrad, pero intenta resolver algunas de sus limitaciones ajustando la tasa de aprendizaje acumulada.
6. Adamax: Es una variante del optimizador Adam que utiliza el promedio móvil infinito en lugar del promedio móvil de segundo orden de los gradientes para actualizar los parámetros.

Anexo de importaciones:
Importaciones

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
```