Understanding Convolutional Neural Networks (CNNs)

1. Basics of Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep neural networks most commonly used in image and video recognition.
They are designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks,
such as convolution layers, pooling layers, and fully connected layers.

Key Components:

- Convolutional Layer: Applies a set of learnable filters to the input image, extracting spatial features.
- Activation Function: Introduces non-linearity (ReLU, Sigmoid, Softmax).
- Pooling Layer: Downsamples feature maps (Max Pooling, Average Pooling, Global Pooling).
- Flatten Layer: Converts multi-dimensional data to 1D before feeding into dense layers.
- Fully Connected Layers: Perform high-level reasoning and classification.
- Dropout: Regularization technique to prevent overfitting by randomly turning off neurons during training.

2. CNN Layer Parameters in PyTorch

Conv2D Parameters:

- in_channels: Number of input channels (e.g. 3 for RGB images)
- out_channels: Number of output channels (filters)
- kernel_size: Size of the filter (e.g. 3x3 or 5x5)
- stride: Step size while applying the filter (default is 1)
- padding: Amount of zero-padding added to preserve output size

BatchNorm2d Parameters:

- num_features: Number of channels to normalize (usually equal to out_channels of Conv2D)

Activation Functions:

- ReLU: f(x) = max(0, x)

- Sigmoid: f(x) = 1 / (1 + exp(-x))
- Softmax: Converts raw scores to class probabilities

Pooling Layers:

- MaxPool2d: Returns the maximum value from each patch of the feature map
- AvgPool2d: Returns the average value from each patch
- AdaptiveAvgPool2d: Outputs a fixed-size (e.g., 1x1) output for any input size

Linear (Fully Connected) Layer Parameters:

- in_features: Size of each input sample
- out_features: Size of each output sample

Dropout Parameters:

- p: Probability of an element to be zeroed (0.5 means 50% dropout)

3. Implementing the CNN Architecture

To implement the custom CNN architecture described:

1. Define a class that extends nn.Module.
2. In __init__, define 3 convolutional blocks, each with Conv2D, BatchNorm, Activation, and Pooling.
3. Add a classification block with Flatten, Dropout, Linear layers, and Softmax at the end.
4. Define the forward() method to sequence the blocks.

Each block should follow this structure:

- Convolution: Extract features with filters
- BatchNorm: Normalize activations
- Activation: ReLU or Sigmoid
- Pooling: Downsample (Max, Average, or Global)
- Fully Connected: Learn final patterns and predict class probabilities

Understanding Convolutional Neural Networks (CNNs)

Use nn.Sequential to group layers, and move model to GPU using .to(device).

Refer to the full implementation in PyTorch for real-world deployment and testing.