1. Simple Convolutional Neural Network (PyTorch Model Explanation)

This section explains the structure and behavior of a simple convolutional neural network implemented using PyTorch.
The model is designed for image classification tasks and follows a standard convolutional neural network architecture.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()

        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(2)

        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(2)

        self.fc1 = nn.Linear(32 * 56 * 56, 64)
        self.fc2 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)

        x = F.relu(self.conv2(x))
        x = self.pool2(x)

        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x
```

Overall Architecture

The model consists of two main stages.
The first stage performs feature extraction using convolution and pooling operations.
The second stage performs classification using fully connected layers.
The input to the model is assumed to be a batch of RGB images with a fixed spatial resolution.

Convolution Layer 1

The first convolution layer processes the raw input image.
It takes an input with three channels, corresponding to the red, green, and blue color components.
This layer produces multiple output feature maps, each representing a different learned visual pattern such as edges or simple textures.
A small convolution kernel is used to focus on local spatial features while preserving the overall structure of the image.
Padding is applied so that the spatial resolution of the output feature maps remains the same as the input at this stage.

Activation Function

After the first convolution, a rectified linear unit activation function is applied.
This activation introduces non-linearity, allowing the model to learn complex patterns beyond linear combinations of input values.
Max Pooling Layer 1

The first pooling layer reduces the spatial resolution of the feature maps.
It downsamples the input by selecting the maximum value within small local regions.
This operation reduces computational complexity and helps the model become more robust to small spatial variations in the input.

Convolution Layer 2

The second convolution layer operates on the output of the first pooling stage.
It increases the number of feature maps, enabling the model to capture more complex and abstract patterns.
At this stage, the input no longer represents raw pixel values but learned features from earlier layers.
The convolution continues to use a small kernel size and preserves spatial alignment through padding.

Activation Function

As in the first convolution stage, a rectified linear unit activation function is applied to introduce non-linearity and improve learning capacity.

Max Pooling Layer 2

The second pooling layer further reduces the spatial resolution of the feature maps.
By this point, the model has transformed the input image into a compact representation of high-level visual features.

Flattening Operation

Before classification, the multi-dimensional feature maps are reshaped into a one-dimensional vector.
This transformation allows the data to be processed by fully connected layers.

Fully Connected Layer 1

The first fully connected layer takes the flattened feature vector and maps it to a lower-dimensional representation.
This layer combines information from all spatial locations and channels to form a global understanding of the input image.
A rectified linear unit activation function is applied to maintain non-linearity.

Fully Connected Layer 2 (Output Layer)

The final fully connected layer produces the output of the model.
Each output element corresponds to a class score.
The number of output units is equal to the number of target classes.
These scores can later be converted into probabilities or used directly for loss computation during training.
Forward Data Flow
During inference or training, data flows sequentially through convolution, activation, pooling, flattening, and fully connected stages.
Each stage progressively transforms the input from raw image data into a high-level representation suitable for classification.
Design Characteristics

This model follows a classic and interpretable CNN design.
It balances simplicity and expressive power, making it suitable for educational purposes, prototyping, and baseline experiments.
The explicit definition of each stage provides clarity and flexibility for modification, extension, or optimization.

2. Simple Convolutional Neural Network (TensorFlow / Keras Model Explanation)

This section explains a simple convolutional neural network implemented using TensorFlow with the Keras API.

```
import tensorflow as tf
from tensorflow.keras import layers, models

def build_model(num_classes=10):
    model = models.Sequential([
        layers.Conv2D(16, (3, 3), padding="same", activation="relu",
                      input_shape=(224, 224, 3)),
        layers.MaxPooling2D(2),

        layers.Conv2D(32, (3, 3), padding="same", activation="relu"),
        layers.MaxPooling2D(2),

        layers.Flatten(),
        layers.Dense(64, activation="relu"),
        layers.Dense(num_classes, activation="softmax")
    ])

    return model
```

The model is designed for image classification tasks and follows a standard convolutional neural network structure.

Overall Architecture

The model is composed of two main functional blocks.
The first block is responsible for extracting visual features from the input images.
The second block performs classification based on the extracted features.
The model assumes an input consisting of color images with a fixed height, width, and three color channels.

First Convolution Stage

The first convolution stage operates directly on the input images.
It applies multiple convolution filters to learn low-level visual features such as edges, color transitions, and simple textures.
Padding is applied so that the spatial resolution of the output feature maps remains consistent with the input resolution at this stage.
An activation function is applied immediately after convolution to introduce non-linearity and improve the model's representational power.

First Pooling Stage

After the first convolution stage, a pooling operation is applied to reduce the spatial resolution of the feature maps.
This operation summarizes local regions by selecting dominant values, which reduces computation and increases robustness to small spatial variations.

Second Convolution Stage

The second convolution stage processes the feature maps produced by the previous pooling layer.
At this point, the input no longer represents raw image pixels but learned feature representations.
The number of feature maps is increased, allowing the model to capture more complex and abstract visual patterns.
As in the first stage, padding preserves spatial alignment, and an activation function introduces non-linearity.

Second Pooling Stage

The second pooling operation further reduces the spatial resolution of the feature maps.
This step compresses the learned features into a more compact representation while retaining the most informative responses.

Flattening Operation

Once feature extraction is complete, the multi-dimensional feature maps are transformed into a one-dimensional feature vector.
This operation prepares the data for processing by fully connected layers.

Fully Connected Layer

The first fully connected layer combines information from all extracted features into a compact representation.
This layer enables the model to learn global relationships between features across the entire image.
An activation function is applied to maintain non-linear decision boundaries.

Output Layer

The final layer produces the classification output of the model.
Each output unit corresponds to a target class.