Convolutional Neural Networks (CNNs): 2D and 1D Variants

An Easy-to-Follow Guide to CNNs for Beginners

Introduction to CNNs as Grid Processors

Think of an image as a grid of squares, like a chessboard but much bigger. Each square (pixel) has a color value. Convolutional Neural Networks (CNNs) look at small patches of this grid to find patterns, such as edges or textures.

- **Grid structure**: Images are 2D grids (height × width) with depth (color channels).
- **Local patches**: CNNs scan small windows (patches) across the grid to detect features.
- **Shared filters**: The same small window of weights (filter) is reused across the entire image, reducing the number of parameters.

This design makes CNNs fast at finding visual patterns and effective for tasks like image classification, object detection, and more.

2D Convolutional Neural Networks (2D CNNs)

How 2D Convolution Works

1. **Filter (Kernel)**: A small 3×3 or 5×5 grid of numbers.
2. **Slide window**: Move the filter over each position in the image grid.
3. **Dot product**: Multiply filter values with the underlying pixel values, then sum up to one number.
4. **Feature map**: The collection of output numbers forms a new grid called a feature map, showing where the filter detected the pattern.

Mathematical formula:

$$(I * K)(i, j) = \sum_{m=1}^{M} \sum_{n=1}^{N} I(i + m - 1, j + n - 1) \times K(m, n)$$

- I: Input image
- K: Kernel of size M×N
- Output size depends on padding and stride

2D CNN Architecture Steps

1. **Input**: H×W×C (Height, Width, Channels) image.Grid shape.
2. **Convolution**: Apply F filters → H'×W'×F feature maps.
3. **Activation**: Apply ReLU (zero out negatives).
4. **Pooling**: Downsample each feature map (e.g., 2×2 max pooling) → smaller grid.
5. **Repeat**: Stack more conv + activation + pooling layers.
6. **Flatten**: Convert final maps to a 1D vector.
7. **Fully Connected**: Dense layers map features to outputs.

Example: 2D CNN in PyTorch

```
import torch
import torch.nn as nn

class Simple2DCNN(nn.Module):
    def __init__(self):
        super().__init__()
        # 1. Convolutional layer: in_channels=3 (RGB), out_channels=16, 3x3 filter
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
        # 2. Pooling layer: 2x2 max pooling
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        # 3. Fully connected layer: input features = 16*16*16, output classes=10
        self.fc1 = nn.Linear(in_features=16*16*16, out_features=10)

    def forward(self, x):
        # Apply convolution -&gt; ReLU -&gt; pooling
        x = self.pool(torch.relu(self.conv1(x)))
        # Flatten the feature maps into a vector
        x = x.view(x.size(0), -1)
        # Fully connected for classification
        x = self.fc1(x)
        return x
```

- **Comments on each line** explain what's happening.
- Assuming 32×32 RGB input, conv1 + pool produce 16×16×16 maps, flattened to 4096 features.

1D Convolutional Neural Networks (1D CNNs)

How 1D Convolution Works

For sequential data (e.g., audio or time series), the grid is one-dimensional: a line of values.

1. **Filter**: A small vector, e.g., length=3.
2. **Slide window**: Move filter along the sequence.
3. **Dot product**: Multiply filter values with sequence segment, sum to one number.

4. **Feature map**: New sequence showing where the filter detected patterns.

Formula:

$$(X * K)(i) = \sum_{m=1}^{M} X(i + m - 1) \times K(m)$$

1D CNN Architecture Steps

1. **Input**: L×C (Length, Channels) sequence.
2. **Conv1D**: Apply F filters → L'×F feature maps.
3. **Activation**: ReLU.
4. **Pooling**: 1D max pooling to shorter sequence.
5. **Repeat**: More conv1D + activation + pooling.
6. **Flatten** or **Global Pool**:
7. **Fully Connected**: Map to output.

Example: 1D CNN in Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, Dense

model = Sequential([
    # 1. Conv1D: 32 filters, kernel size 3, ReLU activation
    Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(100, 1)),
    # 2. MaxPooling1D: pool size 2
    MaxPooling1D(pool_size=2),
    # 3. Global average pooling to get one value per feature map
    GlobalAveragePooling1D(),
    # 4. Dense output layer with sigmoid for binary classification
    Dense(1, activation='sigmoid')
])

# Each line is commented for clarity on its purpose.
```

- **Input shape**: sequences of length 100 with one feature per step.
- **GlobalAveragePooling1D** converts each feature map to a single value, reducing parameters.

Best Practices and Tips

- **Use simple filters**: 3×3 for images, size 3 for sequences is a good starting point.
- **Padding**: Add zeros around edges to keep output size the same.
- **Stride**: Move filter by more than one step for faster downsampling.
- **Pool wisely**: Max pooling retains strongest signals; average pooling smooths noise.
- **Combine with other layers**: For sequences, you can stack 1D CNNs before RNNs or Transformers.

- **Regularize**: Use dropout and batch normalization to avoid overfitting.
- **Transfer learning**: For images, start from pretrained 2D CNNs (ResNet, VGG).

This guide explains CNNs as grid processors in simple language, with thorough comments in code examples to help beginners understand each step.