Module 10: Convolutional Neural Networks

Overview

Convolutional Neural Networks (CNNs) are specialized architectures for processing grid-structured data, especially images. Rather than treating each pixel independently, CNNs exploit spatial structure by learning local patterns that can appear anywhere in the image. This module covers the components of CNNs—convolutions, pooling, and feature maps—and demonstrates how to build image classifiers with Keras.

---

1. Why CNNs for Images?

The Problem with Fully Connected Networks

For a 224×224 RGB image: - Input size: 224 × 224 × 3 = 150,528 features - First dense layer with 1000 neurons: 150 million parameters!

Problems: - Too many parameters □ overfitting - Ignores spatial structure - Not translation invariant

The CNN Solution

CNNs address these by: - **Local connectivity**: Each neuron sees only a small region - **Parameter sharing**: Same filter applied everywhere - **Translation invariance**: Pattern detected regardless of position

---

2. The Convolution Operation

Intuition

A filter (kernel) slides across the image, computing dot products at each position. This detects a specific pattern wherever it appears.

Mathematical View

For a 2D convolution:

```
output[i,j] = Σ Σ input[i+m, j+n] × filter[m, n]
```

The filter is typically 3×3, 5×5, or 7×7.

Example: Edge Detection

A horizontal edge detector:

```
[[-1, -1, -1],
 [ 0,  0,  0],
 [ 1,  1,  1]]
```

This responds strongly to horizontal edges.

Multiple Filters

Each filter learns to detect a different pattern: - Edges at various angles - Corners - Textures - Higher-level features in deeper layers

---

3. Stride and Padding

Stride

How many pixels the filter moves at each step: - Stride 1: Output nearly same size as input - Stride 2: Output half the size

Padding

Adding zeros around the input: - **Valid (no padding)**: Output shrinks - **Same padding**: Output same size as input

Output Size Formula

```
output_size = floor((input_size - filter_size + 2×padding) / stride) + 1
```

---

4. Pooling Layers

Purpose

- Reduce spatial dimensions
- Provide translation invariance
- Reduce computation

Max Pooling

Take the maximum value in each region:

```
[[1, 3],     →  [4]
 [2, 4]]
```

Common: 2×2 pooling with stride 2 □ halves dimensions.

Average Pooling

Take the mean instead of max. Less common except at the end of networks (global average pooling).

---

5. CNN Architecture

Typical Structure

```
Input → [Conv → ReLU → Pool] × N → Flatten → Dense → Output
```

Early Layers

- Small receptive field
- Detect low-level features (edges, colors)
- Many spatial locations

Deeper Layers

- Larger effective receptive field
- Detect high-level features (eyes, wheels, faces)
- Fewer spatial locations, more channels

Example: Simple CNN

```
Conv2D(32, 3×3) → ReLU → MaxPool(2×2)
Conv2D(64, 3×3) → ReLU → MaxPool(2×2)
Conv2D(64, 3×3) → ReLU
Flatten
Dense(64) → ReLU
Dense(num_classes) → Softmax
```

---

6. Famous Architectures

LeNet (1998)

First successful CNN (digit recognition).

AlexNet (2012)

Deep CNN that won ImageNet. Started the deep learning revolution.

VGG (2014)

Very deep with small 3×3 filters. Simple, uniform architecture.

ResNet (2015)

Introduced skip connections. Enabled training very deep networks (100+ layers).

Modern Architectures

- EfficientNet: Compound scaling
- Vision Transformers (ViT): Attention for images

---

7. Transfer Learning

The Idea

Pre-trained models on ImageNet (14M images, 1000 classes) have learned useful features. Reuse these for your task!

Approaches

Feature Extraction: 1. Load pre-trained model (remove top layers) 2. Freeze all weights 3. Add new classification head 4. Train only the new layers

Fine-Tuning: 1. Start with feature extraction 2. Unfreeze some top layers 3. Train with low learning rate

When to Use

- Limited training data (< 10k images)
- Similar domain to ImageNet
- Faster training and better results

---

8. Building CNNs with Keras

Basic CNN

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

Transfer Learning Example

```
base_model = keras.applications.VGG16(weights='imagenet', include_top=False)
base_model.trainable = False

model = keras.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])
```

---

9. Data Augmentation

Why Augment?

More training data = better generalization. Create variations of existing images.

Common Augmentations

- Random rotation
- Random flip (horizontal/vertical)
- Random zoom
- Random translation
- Color jittering

Keras Implementation

```
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1)
])
```

Key Takeaways

1. **CNNs** exploit spatial structure with local filters
2. **Convolutions** detect patterns via sliding filters
3. **Pooling** provides downsampling and translation invariance
4. **Deep CNNs** learn hierarchical features (edges □ parts □ objects)
5. **Transfer learning** reuses pre-trained models effectively
6. **Data augmentation** improves generalization with limited data
7. **Keras** makes building CNNs straightforward

Connections to Other Modules

- **Module 6**: Feedforward networks foundation
- **Module 11**: Regularization for CNNs
- **Module 13**: Vision Transformers as CNN alternative

Further Reading

- [Links to be added]
- Deep Learning Book, Chapter 9: Convolutional Networks
- CS231n: Convolutional Neural Networks for Visual Recognition