DL Demystified 5 - Computer Vision - Convolutional Neural Networks

January 28, 2025

[1]:
```
######################################################################
#                                                                    #
#  CS435 Generative AI: Security, Ethics and Governance              #
#                                                                    #
#  Instructor: Dr. Adnan Masood                                      #
#  Contact:    adnanmasood@gmail.com                                 #
#                                                                    #
#  Notebook is MIT Licensed                                          #
######################################################################
```

1 Convolutional Neural Networks (CNNs)

In this notebook, we'll briefly explore Convolutional Neural Networks (CNNs), with progressive levels of explanation, math, code examples in PyTorch, and a hands-on exercise with TODOs.

1.1 Table of Contents

1.2 1. Building an Intuitive Understanding

Imagine you have a big picture. A CNN helps a computer "look" at small parts of the picture, find important details, and figure out what the picture is about (like identifying if it's a cat or dog).

Think of CNNs like a detective with a magnifying glass. The detective moves the glass over different parts of an image to see if there's anything interesting (like edges, corners, eyes, etc.). By combining information about these smaller parts, the CNN can understand the bigger picture.

A CNN is a specialized type of neural network for analyzing data that has a grid-like structure (e.g., images). It involves applying small filters (kernels) over the image to produce feature maps. By stacking multiple layers, the network learns complex features (from edges to shapes to objects).

CNNs exploit local connectivity (filters only look at local patches) and weight sharing (the same filter is used across different regions). This reduces parameters and allows the model to be more efficient and translation-invariant. Pooling layers (max or average) further reduce spatial dimensions.

Current CNN research extends to advanced architectures (ResNet, DenseNet, Inception) and optimization techniques (BatchNorm, skip connections). Researchers also investigate interpretability (Grad-CAM, saliency maps) to understand how CNNs focus on relevant image regions. Beyond images, CNNs are adapted for 1D signals, 3D volumes, and even graph structures.

1.3 2. Intuition Behind CNNs

Our eyes process the world in local patches. We first recognize edges and simple patterns, then build up to more complex features. CNNs do the same: sliding small filters across images to detect features step by step.

1.4 3. Brief History and Underlying Technology

- **1980 – Fukushima's Neocognitron**: Early concept of hierarchical feature detection.
- **1989 – LeNet (Yann LeCun)**: First popular CNN for handwritten digit recognition.
- **2012 – AlexNet**: Achieved breakthrough in ImageNet classification.
- **2014 onwards**: Deeper architectures (VGG, GoogleNet, ResNet) improved performance.

Key technologies include: - **Convolution Filters**: Shared weights to detect local patterns. - **Backpropagation**: Automated method to learn filter weights. - **High-Performance Hardware (GPUs)**: Speed up matrix operations.

1.5 4. Mathematical Explanation

1.5.1 4.1

1. We take a small window (filter) and slide it over the image.
2. At each position, we multiply the filter values by the overlapping image pixels and add them up.
3. This sum becomes the new "feature" in the output.

1.5.2 4.2 Deeper Dive

In 2D:

$$(I * K)(x, y) = \sum_m \sum_n I(m, n) \cdot K(x - m, y - n)$$

where I is the image, K is the kernel (filter), and (x, y) is the position in the output feature map.

- **Weight**: The values in the filter.
- **Bias**: A constant added after the convolution sum.
- **Activation Function (e.g., ReLU)**: $\text{ReLU}(z) = \max(0, z)$, introducing non-linearity.
- **Pooling**: Reduces the spatial dimension, like picking the maximum value in a small region (max pooling).

1.6 5. Illustrative Example with Code

Below, we'll manually perform a 2D convolution on a small 5x5 image with a 3x3 kernel.

[1]:
```
import torch
import torch.nn as nn
import numpy as np

# 5x5 image
image = torch.tensor([
    [1., 2., 3., 4., 5.],
    [4., 5., 6., 7., 8.],
    [7., 8., 9., 0., 1.],
    [2., 3., 4., 5., 6.],
    [5., 6., 7., 8., 9.]
])

# 3x3 kernel
kernel = torch.tensor([
    [ 1.,  0., -1.],
    [ 1.,  0., -1.],
    [ 1.,  0., -1.]
])

def manual_convolution2d(image, kernel):
    kernel_size = kernel.shape[0]
    output_size = image.shape[0] - kernel_size + 1
    output = torch.zeros((output_size, output_size))
    for i in range(output_size):
        for j in range(output_size):
            region = image[i:i+kernel_size, j:j+kernel_size]
            output[i, j] = torch.sum(region * kernel)
    return output

conv_result = manual_convolution2d(image, kernel)
conv_result
```

[1]:
```
tensor([[-6.,  4.,  4.],
        [-6.,  4.,  4.],
        [-6.,  4.,  4.]])
```

Observe the resulting tensor in `conv_result`. Each entry corresponds to the filter's response in that location.

1.7 6. Step-by-Step Example from Scratch

1. **Choose an input image** (or a batch of images).
2. **Define a filter (kernel)** with some weights.
3. **Convolution**: Multiply and sum over local image patches.

4. **Add bias**.
5. **Apply an activation function** (e.g. ReLU).
6. **Pooling layer** (optional) to reduce spatial dimensions.
7. **Repeat** with more convolution layers.
8. **Flatten** and feed into a fully connected layer if needed.

1.8 7. Illustrative Problem: What Does a CNN Solve?

CNNs are great at tasks like: - **Image Classification**: e.g., recognizing if an image is a cat or a dog. - **Object Detection**: e.g., locating objects in an image. - **Segmentation**: e.g., highlighting which parts of the image are cat vs. background. It's basically giving computers "eyes" to see and interpret images.

1.9 8. Real-World Problem

1.9.1 Example: Medical Imaging

CNNs help in detecting tumors in MRI scans or identifying lesions in chest X-rays. By learning features indicative of certain diseases, a CNN can assist healthcare professionals in diagnosis. ### Other Applications - Self-driving cars (vision-based lane detection) - Face recognition systems - Retail product scanning - Robotics (visual servoing)

1.10 9. Questions to Ponder & Their Answers

1. **Q**: Why do CNNs share weights? **A**: This reduces the number of parameters and helps detect the same feature across different parts of the image.
2. **Q**: Why use ReLU? **A**: ReLU introduces non-linearity, allowing the model to learn more complex patterns.
3. **Q**: How do we handle color images? **A**: Color images have 3 channels (RGB). The filter extends across all channels, so a 3x3 filter becomes 3x3x3.
4. **Q**: What is Padding? **A**: Adding extra rows/columns (often zeros) around the image to maintain the output size or control it.
5. **Q**: How is CNN training performed? **A**: Through backpropagation, computing partial derivatives of the loss w.r.t. each weight in the filters.

1.11 10. A Sample Exercise using PyTorch

We'll train a small CNN on the MNIST dataset (handwritten digits). Some parts are left as TODO.

[3]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

# 1. Load Data
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,)) # mean and std for MNIST
```

```
])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True,␣
 ↪download=True, transform=transform)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False,␣
 ↪download=True, transform=transform)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,␣
 ↪shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64,␣
 ↪shuffle=False)

# 2. Define a Simple CNN Model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # TODO 1: Add a Conv2d layer: in_channels=1, out_channels=8,␣
 ↪kernel_size=3
        # HINT: self.conv1 = nn.Conv2d(1, 8, 3)
        self.conv1 = nn.Conv2d(1, 8, kernel_size=3)

        # TODO 2: Add another Conv2d layer: in_channels=8, out_channels=16,␣
 ↪kernel_size=3
        # HINT: self.conv2 = nn.Conv2d(8, 16, 3)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3)

        # TODO 3: Add a fully connected layer: input features=? output=10 (for␣
 ↪10 digits)
        # HINT: self.fc = nn.Linear(16*some_size, 10)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(16*5*5, 10)

        # # TODO: Example solution: (what is wrong with this?)
        # self.conv1 = nn.Conv2d(1, 8, 3)
        # self.conv2 = nn.Conv2d(8, 16, 3)
        # self.fc = nn.Linear(16*5*5, 10)  # after two conv with kernel=3 each␣
 ↪time, MNIST 28->26->24, then pool or not.

        # TODO - Do you think this would help solve?
        # Can you Calculate the correct input size for the fully connected layer
        # Output of conv2 is (28 - 3 + 1 - 3 + 1) = 24 (without padding)
        # So, the flattened size is 24 * 24 * 16 = ?
        #self.fc = nn.Linear(?, 10) # changed to 24*24*16 which is 9216


    def forward(self, x):
```

```
        # 3. Apply conv1, ReLU, optional pooling
        x = self.conv1(x)
        x = torch.relu(x)
        x = self.pool(x)

        # 4. Apply conv2, ReLU
        x = self.conv2(x)
        x = torch.relu(x)
        x = self.pool(x)

        # 5. Flatten
        x = x.view(x.size(0), -1)

        # 6. Fully connected layer
        x = self.fc(x)
        return x

# 4. Instantiate Model, Define Loss and Optimizer
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 5. Train the Model (1 epoch for demo)
for epoch in range(1):
    model.train()
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch+1}/1], Loss: {loss.item():.4f}")

# 6. Evaluate the Model
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"Test Accuracy: {accuracy:.2f}%")
```

```
Epoch [1/1], Loss: 0.0934
Test Accuracy: 97.28%
```

1.12 11. Glossary

- **Kernel/Filter**: Small matrix of learnable weights used in convolution.
- **Stride**: The step size by which we slide the filter across the image.
- **Padding**: Adding extra rows/columns (often zeros) around the image to control output size.
- **Feature Map**: The output matrix after the convolution of a filter with the image.
- **Pooling**: Operation to reduce spatial dimensions (e.g., max pooling).
- **Fully Connected Layer**: A layer where each neuron connects to all outputs of the previous layer.
- **Activation Function**: A function (like ReLU) that introduces non-linearity.
- **Backpropagation**: The method to compute gradients and update weights.
- **Epoch**: One complete pass through the training dataset.
- **Batch Size**: Number of samples processed before the model updates its parameters.

[4]:
```python
import os, sys, platform, datetime, uuid, socket

def signoff(name="Anonymous"):
    colab_check = "Yes" if 'google.colab' in sys.modules else "No"
    mac_addr = ':'.join(format((uuid.getnode() >> i) & 0xff, '02x') for i in␣
 ↪reversed(range(0, 48, 8)))
    print("+++ Acknowledgement +++")
    print(f"Executed on: {datetime.datetime.now()}")
    print(f"In Google Colab: {colab_check}")
    print(f"System info: {platform.system()} {platform.release()}")
    print(f"Node name: {platform.node()}")
    print(f"MAC address: {mac_addr}")
    try:
        print(f"IP address: {socket.gethostbyname(socket.gethostname())}")
    except:
        print("IP address: Unknown")
    print(f"Signing off, {name}")

signoff("Ali Muhammad Asad")
```

```
+++ Acknowledgement +++
Executed on: 2025-01-28 18:09:57.470399
In Google Colab: No
System info: Linux 6.8.0-51-generic
Node name: alimuhammad-Inspiron-7559
MAC address: 20:47:47:74:94:05
IP address: 127.0.1.1
Signing off, Ali Muhammad Asad
```

[ ]: