ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Лекция 7

Свёрточные нейронные сети (CNN)

Свёртка, фильтры, pooling, LeNet, AlexNet, VGG, ResNet и PyTorch

Преподаватель: Аршын Алтыбай

Кафедра компьютерной инженерии,
Алматинский технологический университет, Алматы, Казахстан
Учебный год: 2026

> **Связь с предыдущими лекциями.** Мы уже знаем, как работают полносвязные слои, обратное распространение ошибки, оптимизаторы и регуляризация. Теперь изучим класс архитектур, который учитывает двумерную структуру изображений и использует *локальные связи* и *совместное использование весов*. Это основа свёрточных нейронных сетей [6, 3, 4].

1. Цель и план лекции

Цель: понять математические и инженерные основы CNN и научиться проектировать сеть для классификации изображений. По завершении лекции студент должен уметь:

1. записывать дискретную двумерную свёртку и отличать её от кросс-корреляции;
2. вручную вычислять выход свёрточного фильтра на небольшой матрице;
3. находить размеры feature maps с учётом padding, stride и dilation;
4. определять число параметров и вычислительную стоимость свёрточного слоя;
5. объяснять pooling, receptive field, эквивариантность сдвигу и ограничения CNN;
6. описывать ключевые идеи LeNet-5, AlexNet, VGG и ResNet;
7. реализовывать и обучать компактную CNN на PyTorch, не допуская утечки тестовых данных;
8. оценивать accuracy, confusion matrix и анализировать ошибки модели.

План: мотивация; тензоры изображений; операция свёртки; геометрия feature maps; многоканальность; pooling; глубокие блоки; архитектуры; обучение и диагностика; практика; вопросы.

2. Почему для изображений недостаточно обычного MLP?

Изображение является не неупорядоченным вектором, а двумерным массивом с сильной *локальной структурой*: соседние пиксели коррелируют, а небольшие объекты (углы, контуры, текстуры) могут появляться в разных местах изображения. Полносвязная сеть, получающая на вход $H \times W \times C$ значений, не учитывает пространственную близость автоматически. Например, для цветного изображения $224 \times 224 \times 3$ и скрытого слоя из 1000 нейронов потребуется

$$(224 \cdot 224 \cdot 3) \cdot 1000 + 1000 = 150\,529\,000$$

параметров только в первом полносвязном слое. Это увеличивает память и риск переобучения [6].

В CNN один и тот же небольшой фильтр применяют к разным пространственным позициям. Такое **разделение параметров** резко уменьшает их число и задаёт полезное индуктивное предположение: обнаружение контура на левом краю изображения должно использовать похожие параметры, что и обнаружение того же контура справа.

> **Главная идея** CNN не «понимает изображение» автоматически: она обучает фильтры на данных. Однако структура сети делает поиск локальных и повторяющихся закономерностей гораздо более экономным, чем плотная матрица связей.

3. Цифровое изображение как тензор

3.1 Пиксели, каналы и пакетная обработка

Пусть $X \in \mathbb{R}^{H \times W \times C}$: H — высота, W — ширина, C — число каналов (для RGB обычно $C = 3$). При обучении данные объединяют в mini-batch. В PyTorch принят порядок `NCHW`: $X \in \mathbb{R}^{N \times C \times H \times W}$; в TensorFlow/Keras часто встречается `NHWC`. Неправильный порядок осей — типичная ошибка новичков [5, 4].

Тип данных	Типичная форма	Смысл
Одно серое изображение	$1 \times 28 \times 28$	Один канал интенсивности
Одно RGB-изображение	$3 \times 64 \times 64$	Три цветовых канала
Пакет RGB	$32 \times 3 \times 64 \times 64$	32 изображения
Feature map	$32 \times 16 \times 30 \times 30$	16 выходных признаковых каналов

3.2 Нормализация интенсивности

Если значения пикселей целочисленные в $[0, 255]$, часто используют $x \mapsto x/255$. Дополнительно применяют стандартизацию по каналам: $x'_c = (x_c - \mu_c)/\sigma_c$. Значения μ_c и σ_c оценивают *по обучающей выборке* либо используют документированные статистики предобученной модели. Не следует подбирать preprocessing, рассматривая тестовые метки [3].

4. Двумерная свёртка и кросс-корреляция

4.1 Математическое определение

Для бесконечных дискретных сигналов математическая свёртка определяется с переворотом ядра:

$$(X * K)[i, j] = \sum_{a} \sum_{b} X[i - a, j - b] K[a, b]. \tag{1}$$

В большинстве библиотек глубокого обучения операция под названием `Conv2d` фактически вычисляет **кросс-корреляцию** (без переворота ядра):

$$Y[i, j] = \sum_{a=0}^{k_H - 1} \sum_{b=0}^{k_W - 1} K[a, b] X[i + a, j + b] + b_0. \tag{2}$$

Поскольку элементы K обучаются, различие в соглашении обычно не ограничивает выразительность архитектуры, но важно в ручных вычислениях и при сравнении библиотек [6].

4.2 Ручной численный пример

Рассмотрим вход и фильтр:

$$X = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 3 & 1 & 2 & 0 \\ 0 & 1 & 3 & 2 \\ 2 & 0 & 1 & 1 \end{pmatrix}, \quad K = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}.$$

Выберем *valid* кросс-корреляцию, шаг $S = 1$, без bias. Размер результата 3×3. Первый элемент:

$$Y_{1,1} = 1 \cdot 1 + 2 \cdot 0 + 3(-1) + 1 \cdot 1 = -1.$$

Аналогично получаем

$$Y = \begin{pmatrix} -1 & 3 & -2 \\ 4 & 3 & 1 \\ -2 & 2 & 3 \end{pmatrix}. \tag{3}$$

Важно: формула (3) использует *кросс-корреляцию*, а не математическую свёртку с перевёрнутым фильтром. При ручной проверке всегда фиксируйте соглашение.

Пример 1. Ручная кросс-корреляция на NumPy

```python
import numpy as np
X = np.array([[1, 2, 0, 1], [3, 1, 2, 0],
              [0, 1, 3, 2], [2, 0, 1, 1]])
K = np.array([[1, 0], [-1, 1]])
H, W = X.shape
kh, kw = K.shape
Y = np.zeros((H - kh + 1, W - kw + 1), dtype=int)
for i in range(Y.shape[0]):
    for j in range(Y.shape[1]):
        Y[i, j] = np.sum(X[i:i+kh, j:j+kw] * K)
print(Y)
```

Ожидаемый вывод

```
[[-1  3 -2]
 [ 4  3  1]
 [-2  2  3]]
```

4.3 Что выявляют обучаемые фильтры?

На ранних слоях CNN фильтры могут реагировать на простые структуры: горизонтальные и вертикальные границы, углы, цветовые перепады, текстуры. На последующих слоях комбинации локальных признаков образуют более сложные структуры. При этом заранее заданные детекторы границ — удобная иллюстрация, *не обязательные неизменные ядра* обучаемой CNN [4].

5. Padding, stride и dilation: размер выходной карты

Пусть высота входа H, размер ядра K_H, padding P_H с каждой стороны, stride S_H, dilation D_H. Эффективный размер ядра:

$$K_{\text{eff},H} = D_H(K_H - 1) + 1.$$

При стандартных целочисленных параметрах высота выхода:

$$H_{\text{out}} = \left\lfloor \frac{H + 2P_H - D_H(K_H - 1) - 1}{S_H} + 1 \right\rfloor, \tag{4}$$

и аналогично для W_{out}. Это выражение является основой проверки размерностей CNN. Данные параметры вычисляются отдельно для каждой пространственной координаты.

5.1 Примеры

1. $H = 32$, $K_H = 3$, $P_H = 0$, $S_H = 1$, $D_H = 1$: $H_{\text{out}} = 30$.
2. $H = 32$, $K_H = 3$, $P_H = 1$, $S_H = 1$: $H_{\text{out}} = 32$ (сохранение размера при нечётном ядре и stride 1).
3. $H = 32$, $K_H = 3$, $P_H = 1$, $S_H = 2$: $H_{\text{out}} = 16$.
4. $H = 15$, $K_H = 3$, $P_H = 0$, $D_H = 2$, $S_H = 1$: $K_{\text{eff}} = 5$, поэтому $H_{\text{out}} = 11$.

Различайте `padding='same'` как соглашение конкретного фреймворка и точное равенство размерностей: поведение при stride больше 1 зависит от реализации.

6. Многоканальная свёртка

Для входа $X \in \mathbb{R}^{C_{\text{in}} \times H \times W}$ и C_{out} фильтров тензор весов имеет размер

$$K \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_H \times k_W}.$$

Выходной канал o равен

$$Y_{o,i,j} = b_o + \sum_{c=1}^{C_{\text{in}}} \sum_{a=0}^{k_H-1} \sum_{b=0}^{k_W-1} K_{o,c,a,b}\, X_{c,iS_H+a-P_H,jS_W+b-P_W}, \tag{5}$$

где индексы для padding трактуются с соответствующим расширением входа, а для краткости dilation в записи опущен.

6.1 Число обучаемых параметров

В обычной свёртке с bias:

$$N_{\text{par}} = C_{\text{out}} \left(C_{\text{in}} k_H k_W + 1\right). \tag{6}$$

Например, RGB-вход, 16 фильтров 3×3: $16(3 \cdot 3 \cdot 3 + 1) = 448$ параметров. Независимо от того, имеет изображение пространственный размер 32×32 или 512×512, число *параметров* неизменно (но число операций и объём feature maps существенно возрастают).

6.2 Оценка вычислительной стоимости

Для стандартной свёртки количество умножений порядка

$$H_{\text{out}} W_{\text{out}} C_{\text{out}} C_{\text{in}} k_H k_W.$$

При сравнении вычислительных требований важно оговаривать, считают ли MACs как одну операцию или как две FLOPs. Также нужно учитывать память для активаций при обучении.

7. Локальные связи, эквивариантность и receptive field

Локальная связность: выходной нейрон зависит от ограниченной области входа. **Совместное использование весов:** один фильтр применяется к разным позициям. **Эквивариантность сдвигу:** для идеализированной свёртки (без граничных эффектов и изменения разрешения) сдвиг входа приводит к соответствующему сдвигу выхода. Это *не* то же самое, что инвариантность: классификатор может приблизительно сохранять ответ на небольшие сдвиги после pooling/augmentation, но абсолютной гарантии нет [6].

Для цепочки свёрток/пулинга receptive field рассчитывается рекурсивно. Пусть r_l — размер receptive field, j_l — шаг по исходному изображению между соседними выходными точками. Для слоя с эффективным ядром k_l^{eff} и stride s_l:

$$\boxed{j_l = j_{l-1} s_l, \qquad r_l = r_{l-1} + (k_l^{\text{eff}} - 1) j_{l-1}}, \quad r_0 = j_0 = 1.$$

Например, две свёртки 3×3 со stride 1 дают $r_1 = 3$, $r_2 = 5$. После pooling 2×2 со stride 2 получаем $r_3 = 6$, $j_3 = 2$. Следующая свёртка 3×3 увеличит поле до $r_4 = 10$. Теоретическое поле восприятия отличается от *effective receptive field* по распределению фактического влияния пикселей.

8. Pooling и субдискретизация

Pooling уменьшает пространственные размеры и объединяет локальную информацию. Для блока из четырёх значений

$$\begin{pmatrix} 1 & 5 \\ 3 & 2 \end{pmatrix}$$

max-pooling возвращает 5, average-pooling — $(1+5+3+2)/4 = 2.75$. При непересекрывающихся окнах 2×2 со stride 2 карта 28×28 превращается в 14×14.

Операция	Преимущества	Ограничения
Max pooling	Сохраняет сильнейший локальный отклик	Может отбрасывать слабые, но полезные детали
Average pooling	Усредняет информацию, сглаживает ответ	Снижает контраст локальных максимумов
Strided convolution	Понижение разрешения с обучаемыми параметрами	Возможны aliasing и потеря мелких объектов
Global average pooling	Даёт по числу на выходной канал	Теряется точная пространственная локализация

> Pooling обычно не имеет обучаемых параметров, но не является бесплатным с точки зрения вычислений и потери пространственной информации. Для сегментации и локализации часто нужны специальные способы восстановления разрешения.

9. Типичная структура CNN и поэтапный расчёт формы

Рассмотрим классификатор для $1 \times 28 \times 28$ изображения:

Input $\to$ Conv$(1 \to 8, 3, p = 1)$ $\to$ ReLU $\to$ MaxPool(2) $\to$ Conv$(8 \to 16, 3, p = 1)$ $\to$ ReLU $\to$ Ma

Формы и параметры:

Слой	Размер (без batch)	Параметры
Input	$1 \times 28 \times 28$	0
Conv $1 \to 8$, 3×3, $p = 1$	$8 \times 28 \times 28$	$8(9 + 1) = 80$
MaxPool 2×2	$8 \times 14 \times 14$	0
Conv $8 \to 16$, 3×3, $p = 1$	$16 \times 14 \times 14$	$16(8 \cdot 9 + 1) = 1168$
MaxPool 2×2	$16 \times 7 \times 7$	0
Flatten	784	0
Linear $784 \to 10$	10	$784 \cdot 10 + 10 = 7850$
Итого		**9098**

Это существенно меньше числа параметров полносвязного слоя из примера в начале лекции. Но сравнение само по себе не является доказательством лучшего качества: оно зависит от задачи, архитектуры и данных.

10. Нелинейность, нормализация и обучение CNN

Простейший блок имеет вид

$$Z^{(l)} = \mathrm{Conv}(A^{(l-1)}; W^{(l)}, b^{(l)}), \qquad A^{(l)} = \mathrm{ReLU}(Z^{(l)}).$$

Часто используется порядок `Conv--BatchNorm--ReLU`, хотя существуют и pre-activation конструкции. Для классификации из K классов последний слой выдаёт logits $z \in \mathbb{R}^K$, а вероятности могут быть вычислены через softmax:

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$

Кросс-энтропийная потеря для класса y:

$$\ell(z, y) = -\log p_y = -z_y + \log \sum_j e^{z_j}.$$

В `torch.nn.CrossEntropyLoss` передают *logits, не softmax-вероятности*. Фреймворк вычисляет устойчивую комбинацию log-softmax и NLL внутри [5].

Для свёрточных весов производные $\partial \ell / \partial K$ вычисляются цепным правилом; каждый элемент ядра получает сумму вкладов от всех позиций, где он применялся. Это и отражает *совместное использование весов* во время backpropagation [7, 6].

11. Классические архитектуры CNN

11.1 LeNet-5 (1998)

LeNet-5 (Y. LeCun и соавторы) — исторически важная архитектура для распознавания рукописных символов. Она использует последовательность свёрточных и subsampling-слоёв и классифицирующую часть. Это показало, как локальные фильтры и разделяемые веса можно обучать end-to-end [8, 6].

11.2 AlexNet (2012)

AlexNet (A. Krizhevsky, I. Sutskever, G. Hinton) продемонстрировала преимущества глубоких CNN на ImageNet. Среди важных компонентов — обучение на GPU, ReLU, data augmentation, Dropout и крупномасштабная оптимизация [9]. Не следует воспринимать исходную AlexNet как рекомендуемый вариант для каждой современной задачи: её ценность прежде всего историческая и методологическая.

11.3 VGG (2014)

VGG (K. Simonyan, A. Zisserman) исследовала глубину и использование последовательностей небольших 3×3 фильтров. Две свёртки 3×3 имеют теоретическое receptive field 5×5 (при stride 1), добавляя нелинейность между слоями. При равных числах каналов C две такие свёртки используют порядка $18C^2$ весов против $25C^2$ у одной 5×5 (без учёта bias), хотя это сопоставление не учитывает промежуточные активации и конкретные входные/выходные каналы [10].

11.4 ResNet (2015/2016)

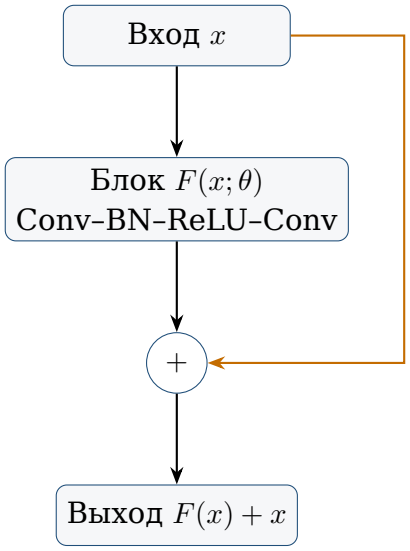
ResNet (K. He и соавторы) вводит **остаточное соединение**:

$$y = F(x;\theta) + x. \tag{7}$$

Если формы $F(x)$ и x не совпадают, используют согласующее преобразование shortcut, например 1×1 свёртку с нужным stride. Для простого остаточного блока

$$\frac{\partial y}{\partial x} = \frac{\partial F}{\partial x} + I,$$

то есть в градиенте возникает прямой путь через тождественное отображение. Это помогает оптимизировать глубокие сети, но не означает, что градиент гарантированно никогда не затухает [11].



Архитектура	Главная идея	Практический урок
LeNet-5	Раннее end-to-end обучение локальных фильтров	Свёртки полезны для изображений
AlexNet	Глубокое CNN и масштабное GPU-обучение	Данные, GPU и оптимизация имеют значение
VGG	Повторяющиеся малые фильтры и глубина	Слои можно строить из простых блоков
ResNet	Residual connections	Обучение глубоких сетей легче с shortcut-путями

12. Современные инженерные приёмы

12.1 Transfer learning

Если данных немного, разумно использовать предобученную сеть (например, ResNet18 или MobileNet) в качестве feature extractor. Сначала можно заморозить backbone и обучить классификационную голову, затем аккуратно разморозить часть слоёв и применить небольшой learning rate. Входная нормализация должна соответствовать требованиям выбранных предобученных весов [3, 4, 5].

12.2 Data augmentation

Горизонтальное отражение, crop, небольшие изменения яркости и повороты могут увеличивать разнообразие обучающих примеров. Но преобразования должны сохранять метку для *конкретной задачи*: отражение фотографии кошки обычно допустимо, а отражение некоторых дорожных знаков или медицинских снимков может изменить смысл.

12.3 Эффективные свёртки

Depthwise separable convolution разделяет пространственную обработку каналов и их смешивание 1×1 свёрткой. При равном числе выходных каналов это может уменьшить параметры относительно стандартной $k \times k$ свёртки: приблизительно $C_{\text{in}}k^2 + C_{\text{in}}C_{\text{out}}$ вместо $C_{\text{in}}C_{\text{out}}k^2$. Реальная скорость зависит от аппаратуры и реализации.

13. Практическая часть I: формы тензоров в PyTorch

Пример 2. Проверка размерностей и числа параметров

```python
import torch
from torch import nn

conv = nn.Conv2d(in_channels=3, out_channels=16,
                 kernel_size=3, stride=2, padding=1)
x = torch.randn(8, 3, 64, 64)
y = conv(x)
print(tuple(y.shape))
print(sum(p.numel() for p in conv.parameters()))
```

Ожидаемый вывод

```
(8, 16, 32, 32)
448
```

Числа совпадают с формулами (4) и (6). Для debugging полезно печатать формы на каждом шаге forward pass.

14. Практическая часть II: обучение небольшой CNN

Ниже приведён воспроизводимый пример классификации *синтетических* изображений размером 16×16: класс 0 содержит вертикальную яркую линию, класс 1 — горизонтальную. Это демонстрация архитектуры и цикла обучения; высокое качество на искусственных данных не доказывает пригодность к реальным изображениям.

Пример 3. Подготовка синтетических изображений

```python
import numpy as np
import torch
from torch import nn
from torch.utils.data import TensorDataset, DataLoader

torch.manual_seed(42)
rng = np.random.default_rng(42)
N = 800
X = rng.normal(0, 0.12, size=(N, 1, 16, 16)).astype('float32')
y = np.zeros(N, dtype='int64')
```

Пример 3 (продолжение)

```
for i in range(N):
    cls = i % 2
    y[i] = cls
    pos = rng.integers(4, 12)
    if cls == 0:
        X[i, 0, :, pos] += 1.0  # vertical line
    else:
        X[i, 0, pos, :] += 1.0  # horizontal line

perm = rng.permutation(N)
tr, va, te = perm[:560], perm[560:680], perm[680:]
X = torch.from_numpy(X)
y = torch.from_numpy(y)
train_dl = DataLoader(TensorDataset(X[tr], y[tr]),
                      batch_size=32, shuffle=True)
val_dl = DataLoader(TensorDataset(X[va], y[va]),
                    batch_size=128)
test_dl = DataLoader(TensorDataset(X[te], y[te]),
                     batch_size=128)
print(len(tr), len(va), len(te))
```

Ожидаемый вывод

```
560 120 120
```

Пример 4. Архитектура CNN и цикл обучения

```
class TinyCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 8, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(8, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((1, 1)),
        )
        self.classifier = nn.Linear(16, 2)

    def forward(self, x):
        return self.classifier(self.features(x).flatten(1))

model = TinyCNN()
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.005)

def evaluate(loader):
    model.eval()
    correct = total = 0
```

Пример 4 (продолжение)

```
    with torch.no_grad():
        for xb, yb in loader:
            pred = model(xb).argmax(dim=1)
            correct += (pred == yb).sum().item()
            total += len(yb)
    return correct / total

best_val = -1
for epoch in range(15):
    model.train()
    for xb, yb in train_dl:
        optimizer.zero_grad()
        logits = model(xb)
        loss = loss_fn(logits, yb)
        loss.backward()
        optimizer.step()
    val_acc = evaluate(val_dl)
    if val_acc > best_val:
        best_val = val_acc
        best_state = {k: v.detach().clone()
                      for k, v in model.state_dict().items()}
    print(f'epoch {epoch+1:2d}: validation={val_acc:.3f}')

model.load_state_dict(best_state)
print('test accuracy:', round(evaluate(test_dl), 3))
```

В отличие от примеров с заранее указанным выводом, точные метрики обучения здесь зависят от вычислительной среды; **не следует подставлять вымышленные значения accuracy**. Запускайте код и фиксируйте фактически полученные метрики. Тестовая выборка используется лишь после выбора модели по validation.

15. Практическая часть III: анализ ошибок

Accuracy — доля верных ответов, но она может скрывать проблемы при дисбалансе классов. Для оценки реального классификатора дополнительно применяют confusion matrix, precision, recall, F1 и анализ ошибочных изображений [3]. Для чувствительных приложений важно оценивать качество на отдельных группах и при изменении условий съёмки.

Пример 5. Матрица ошибок на отложенной выборке

```
from sklearn.metrics import confusion_matrix, classification_report

model.eval()
all_pred, all_true = [], []
with torch.no_grad():
```

Пример 5 (продолжение)

```
    for xb, yb in test_dl:
        all_pred.extend(model(xb).argmax(1).tolist())
        all_true.extend(yb.tolist())
print(confusion_matrix(all_true, all_pred))
print(classification_report(all_true, all_pred,
                            target_names=['vertical','horizontal'],
                            zero_division=0))
```

В итоговой демонстрации студент должен представить не только общую accuracy, но и конкретные ошибки, их вероятные причины и ограничения используемого набора данных.

16. Ошибки при проектировании CNN

1. **Неправильная форма входа:** путаница между NCHW и NHWC.
2. **Неверный размер Flatten:** не учтены pooling, stride или padding.
3. **Лишний Softmax:** применение softmax перед `CrossEntropyLoss`.
4. **Отсутствие `model.eval()`:** Dropout/BatchNorm работают в режиме обучения на этапе проверки.
5. **Data leakage:** использование связанных изображений из одного объекта в train и test без группировки.
6. **Неподходящая augmentation:** преобразование разрушает метку.
7. **Недостаточный error analysis:** оценка только по train accuracy.
8. **Нереалистичный benchmark:** отсутствие сравнения с простым baseline и замера времени inference.

17. Краткие выводы

CNN используют локальные фильтры и совместное использование весов для эффективной обработки изображений. Геометрия каждого слоя определяется ядром, padding, stride и dilation; число параметров — количеством входных и выходных каналов и размером ядра. Pooling и strided convolutions уменьшают разрешение. LeNet, AlexNet, VGG и ResNet демонстрируют важнейшие этапы развития CNN: от локальных фильтров до очень глубоких остаточных архитектур. Однако качество определяется также данными, протоколом оценки, регуляризацией и инженерными решениями.

18. Контрольные вопросы

1. Почему полносвязные сети неэффективны для больших изображений?
2. Что означает локальная связность в CNN?
3. Объясните совместное использование весов.
4. В каком виде PyTorch хранит пакет цветных изображений?
5. Запишите формулу двумерной кросс-корреляции.
6. Чем кросс-корреляция отличается от математической свёртки?
7. Как вычисляется выходная высота свёрточного слоя?
8. Что означают padding, stride и dilation?
9. Вычислите высоту выхода для $H = 64, K = 5, P = 2, S = 2, D = 1$.
10. Запишите формулу числа параметров Conv2d с bias.
11. Сколько параметров у Conv2d $3 \to 32$ с ядром 5×5?
12. Почему число параметров свёртки не зависит от H и W?
13. Чем feature map отличается от исходного изображения?
14. Что такое receptive field?
15. Почему два слоя 3×3 не эквивалентны одному 5×5 во всех отношениях?
16. Чем max pooling отличается от average pooling?
17. Что означает эквивариантность относительно сдвига?
18. Является ли CNN строго инвариантной к любому сдвигу?
19. Опишите основные блоки LeNet-5.
20. Какие идеи сделали AlexNet важным этапом развития CNN?
21. Почему VGG использует последовательности 3×3 свёрток?
22. Запишите формулу остаточного блока ResNet.
23. Зачем в residual shortcut иногда нужна 1×1 свёртка?
24. Почему не нужно применять softmax перед `CrossEntropyLoss`?
25. Какие показатели кроме accuracy нужны при оценке CNN?

19. Задания для самостоятельной работы

1. Вручную вычислите кросс-корреляцию матрицы 3×3 с ядром 2×2, затем проверьте NumPy.
2. Выведите формулу (4) из геометрии перемещения окна.
3. Для входа 64×64 последовательно вычислите размеры после Conv($5, p = 2, s = 2$), MaxPool($2, s = 2$) и Conv($3, p = 1, s = 1$).
4. Для сети Conv($3 \to 16, k = 3$), Conv($16 \to 32, k = 3$), Linear($32 \cdot 8 \cdot 8 \to 10$) подсчитайте параметры при наличии bias.

5. Напишите собственную реализацию valid кросс-корреляции в NumPy и сравните с PyTorch для одноканального случая.
6. Сравните max pooling и average pooling на искусственных матрицах с выбросами.
7. Реализуйте CNN из лекции на синтетических данных и сохраните фактическую матрицу ошибок.
8. Обучите два варианта сети: с одним и двумя свёрточными блоками; сравните validation accuracy и число параметров.
9. Изучите код residual block и реализуйте блок с тождественным shortcut.
10. Проведите мини-исследование transfer learning на небольшом открытом наборе изображений; опишите источник, лицензию и ограничения данных.

20. Литература

В основу материала положены теоретические объяснения [6, 7], практические учебники [3, 4, 5] и вводные издания [1, 2]. Исторические CNN-архитектуры представлены по оригинальным публикациям [8, 9, 10, 11].

Список литературы

[1] Trask A. W. *Grokking Deep Learning*. Manning Publications, 2019. Русское издание: «Грокаем глубокое обучение». https://www.manning.com/books/grokking-deep-learning.

[2] Rashid T. *Make Your Own Neural Network*. 2016. Русское издание: «Глубокое обучение. Лёгкая теория и практика на Python». https://makeyourownneuralnetwork.blogspot.com/.

[3] Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed. O'Reilly Media, 2022. https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/.

[4] Chollet F. *Deep Learning with Python*. 2nd ed. Manning Publications, 2021. https://www.manning.com/books/deep-learning-with-python-second-edition.

[5] Stevens E., Antiga L., Viehmann T. *Deep Learning with PyTorch*. Manning Publications, 2020. https://www.manning.com/books/deep-learning-with-pytorch.

[6] Goodfellow I., Bengio Y., Courville A. *Deep Learning*. MIT Press, 2016. https://www.deeplearningbook.org/.

[7] Nielsen M. A. *Neural Networks and Deep Learning*. 2015. http://neuralnetworksanddeeplearning.com/.

[8] LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 1998, 2278–2324. https://doi.org/10.1109/5.726791.

[9] Krizhevsky A., Sutskever I., Hinton G. ImageNet Classification with Deep Convolutional Neural Networks. *NeurIPS*, 2012. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[10] Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*, 2015. https://arxiv.org/abs/1409.1556.

[11] He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. *CVPR*, 2016, 770–778. https://doi.org/10.1109/CVPR.2016.90.