《神经网络与深度学习》
实验报告

实 验 题 目 验 证 码 识 别

学 院 系 别 信 息 学 院

专 业 名 称 人 工 智 能 系

学 生 姓 名 徐 勉

学 生 学 号 32320222202791

任 课 教 师 曹 东 林

2025 年 5 月 19 日

目录

一、 实验目的

（1）设计实现一种可实现验证码识别的深度学习算法。

（2）基于已经给定的 train.zip 文件内部的数据集对模型进行训练。

（3）设计实现图像的批量识别，给定目录，识别目录中的所有验证码。

二、 实验环境

操作系统：Ubuntu 22.04.5 LTS

深度学习框架：PyTorch

GPU：RTX 4090

CUDA：12.3

三、 实验方法

3.1 实验设置

3.1.1 数据集特点

1000 张验证码图片

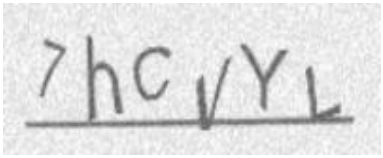


图 1 验证码图像

字符数量：6 位

字符范围：[a-z]，[A-Z]，[0-9]

干扰线：随机，可能是曲线，可能穿过字符

噪声：高斯噪声

文件名：label_编号.png

3.1.2 实验背景

验证码识别的本质是文字识别，从自然场景图片中进行文字识别，需要包括 2 个步骤：

文字检测：解决的问题是哪里有文字，文字的范围有多少（目标检测）。

文字识别：对定位好的文字区域进行识别，主要解决的问题是每个文字是什么，将图像中的文字区域进转化为字符信息[1]。

图 2 文字识别

当前，在验证码识别领域，CRNN（Convolutional Recurrent Neural Network，卷积循环神经网络）结合 CTCLoss（Connectionist Temporal Classification Loss，连接时序分类损失函数）是一种较为经典且广泛应用的模型架构。CRNN 通过 CNN 部分提取验证码图像的特征，再利用 RNN 部分对特征序列进行建模，最终通过 CTCLoss 解决序列预测中输入输出长度不一致的问题，实现对验证码字符序列的识别。

与此同时，YOLO（You Only Look Once）系列模型作为目标检测领域的代表性算法，以其高效的“单次检测”特性在众多目标检测任务中表现出色。YOLO 模型将目标检测任务转化为一个单一的回归问题，能够直接预测目标物体的边界框位置和类别信息，具有实时性强、对目标位置变化和背景干扰具有一定鲁棒性等优势。鉴于验证码识别本质上也可以看作是对验证码中各个字符目标的定位与分类任务，将 YOLO 模型引入验证码识别领域具有潜在的研究价值。

3.2 Baseline 模型（CRNN + CTCLoss）

3.2.1 模型架构

CRNN 模型，即将 CNN 与 RNN 网络结合，共同训练。主要用于在一定程度上实现端到端（end-to-end）地对不定长的文本序列进行识别，不用先对单个文字

进行切割，而是将文本识别转化为时序依赖的序列学习问题，即基于图像的序列识别[2]。

结构	描述
CNN（卷积层）：	使用深度 CNN，对输入图像提取特征，得到特征图；
RNN（循环层）：	使用 双向 RNN（BLSTM）对特征序列进行预测，对序列中的每个特征向量进行学习，并输出预测标签分布；
CTC loss（转录层）：	把从循环层获取的标签分布转换成最终的标签序列。

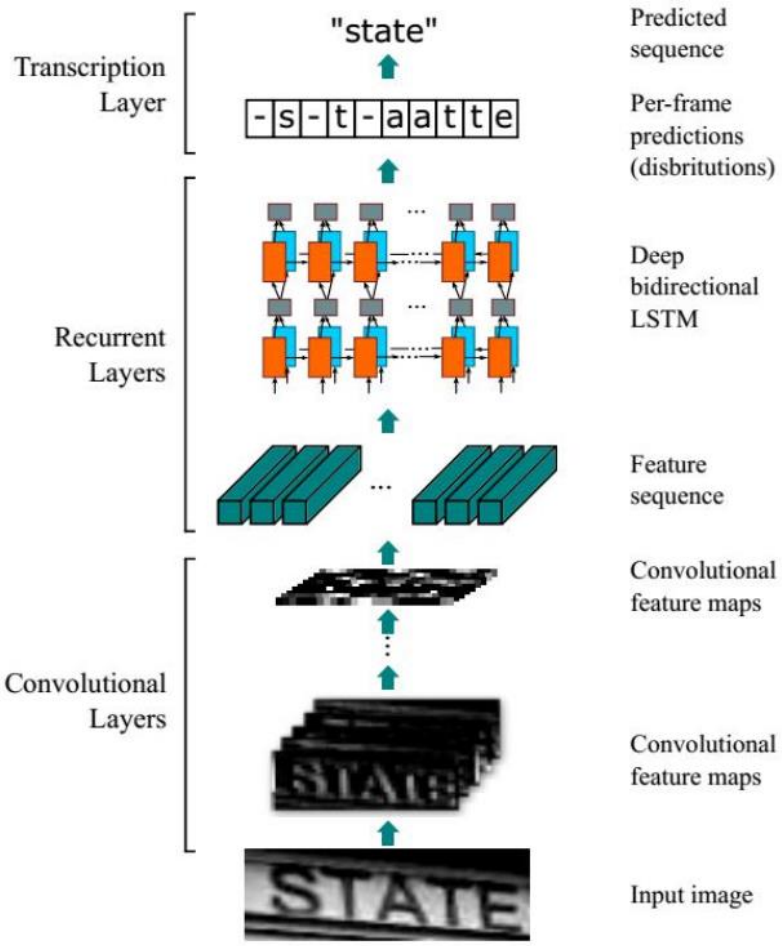


图 3 CRNN 模型架构

3.2.2 模型优缺点

（1）优势：

①擅长序列识别：CRNN 结合了卷积神经网络（CNN）和循环神经网络（RNN）的优点，特别适用于序列识别任务，如验证码中的字符序列识别。它能够通过卷积层捕捉图像局部特征，然后利用循环层处理序列信息，实现准确的文字识别。

②处理不定长文本：CRNN 通过引入 CTC（Connectionist Temporal Classification）Loss，能够处理不定长的文本序列，适用于各种复杂场景下的验证码识别。

（2）劣势：

①序列依赖性：CRNN 假设字符按固定顺序排列，若验证码中字符顺序无规律（如随机打乱），CRNN 需依赖额外规则修正，而 YOLO 无此限制。

②CRNN 的数据增强通常局限于图像变换（如旋转、仿射），难以生成符合验证码逻辑的合成数据。YOLO 可通过 Mosaic 增强（拼接多张图像）、Copy-Paste（随机粘贴字符）等方式模拟复杂验证码场景，提升模型对罕见组合的识别能力。

3.3 改进模型（YOLO）

3.3.1 模型架构

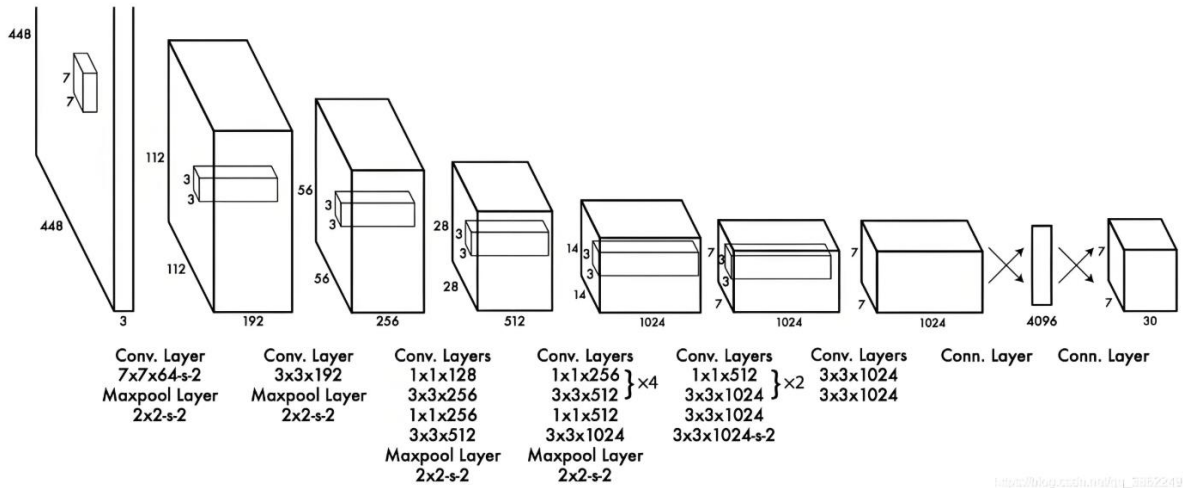


图 4 YOLOv1 模型架构

（1）主干网络（Backbone）

功能：从输入图像中提取特征，采用轻量化设计减少计算量和参数，同时保持较强特征提取能力。

结构：

①初始卷积层：使用常规卷积操作（如 Conv 模块）进行初步特征提取，逐步增加特征图通道数并调整尺寸。

②C3k2 模块：继承自 C2f 模块，通过 c3k 参数决定使用 C3k 或 Bottleneck 结构，堆叠多个 C3k2 模块构建深层网络以提取更丰富特征。

（2）颈部结构（Neck）

功能：融合和传递主干网络提取的特征，供检测头进行准确预测。

结构：

①SPPF 模块：以不同尺度汇集图像区域特征，提高捕获不同大小物体（尤其是小物体）的能力。

②C2PSA 模块：引入注意力机制，强调特征图中的空间相关性，提升对重要区域（如小或部分遮挡对象）的关注度。

上采样与拼接：通过上采样调整特征图尺寸，并通过拼接融合不同尺度特征图，得到更丰富的特征表示。

（3）检测头（Head）

功能：根据颈部结构传递的特征图进行预测，输出目标的边界框坐标、类别概率和置信度等信息。

结构：

①多尺度预测：从三个不同尺度的特征图（P3、P4、P5）输出预测，分别对应图像中的不同粒度级别，确保小物体被精细检测，大物体被高级特征捕获。

②预测模块：在每个尺度的特征图上使用卷积层进行预测，并通过非极大值抑制（NMS）算法筛选和优化预测结果，得到最终的目标检测结果[3]。

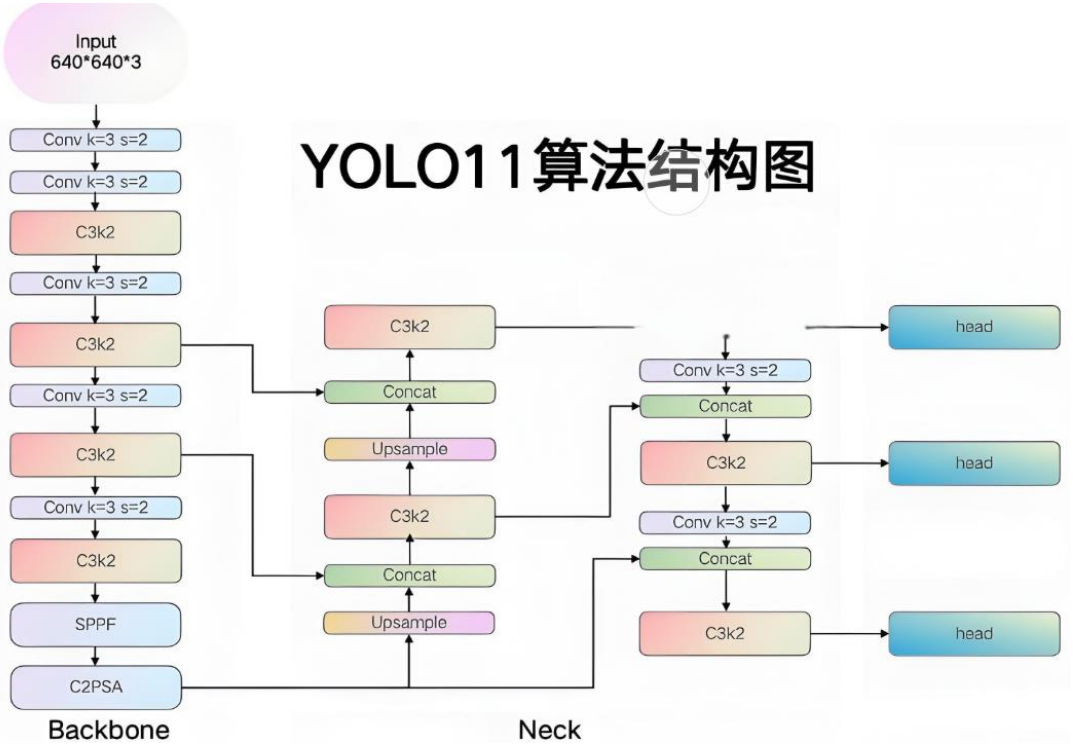


图 5 YOLO11 结构

3.3.2 模型优缺点

（1）优势：

①检测速度快：YOLO 是一种实时目标检测算法，能够快速定位图像中的目标物体，包括验证码中的字符或图形元素。这使得 YOLO 在验证码识别中具有较高的处理效率。

②全局视野：YOLO 在预测时利用了整幅图像的信息，可以更好地利用上下文信息来预测边界框和类别，有助于准确识别验证码中的复杂图案或字符组合。

③泛化能力强：YOLO 直接学习图像的表示，因此可以更容易地泛化到新的、未见过的验证码类型。

（2）劣势：

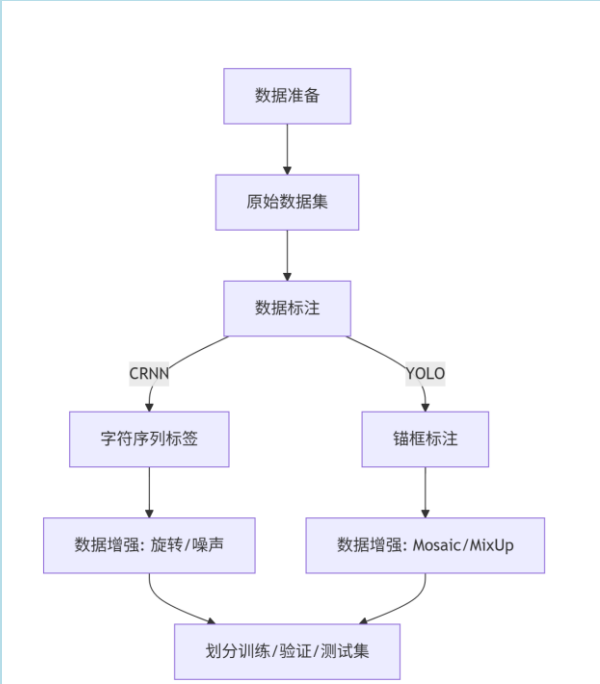
①小目标检测效果差：在验证码识别中，如果字符或图形元素较小，YOLO 的检测效果可能会受到影响，容易出现漏检或定位不准确的情况。

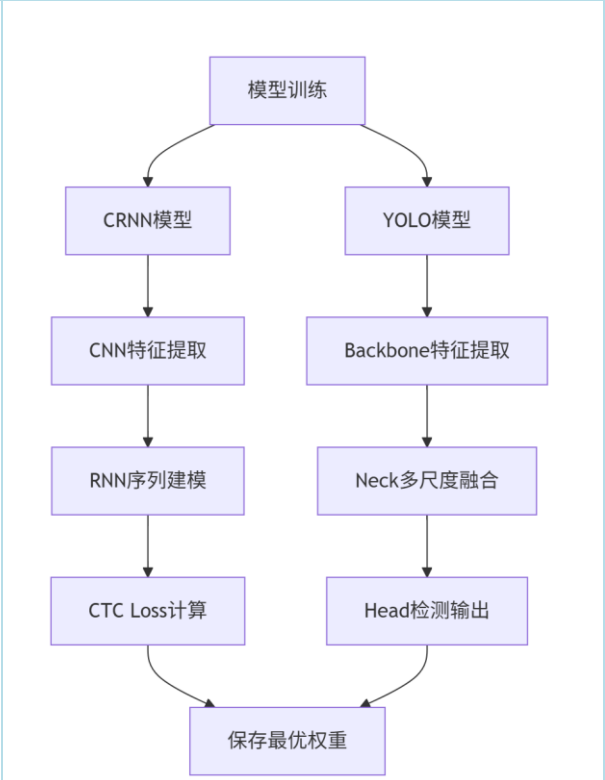
②定位误差：对于重叠或密集排列的验证码元素，YOLO 可能会出现定位误差，导致识别结果不准确。

四、 实验步骤

实验流程图：

数据准备	**模型训练**





4.1 模型定义

4.1.1 CRNN 模型

（1）主干网络 CNN

①功能：提取输入图像的局部特征，生成高维特征图[4]。

②结构：

输入：RGB 图像（尺寸调整为 (80, 200)）。

Conv2d(3, 32, kernel_size=3, padding=1) + ReLU + MaxPool2d(2,2)

Conv2d(32, 64, kernel_size=3, padding=1) + ReLU + MaxPool2d(2,2)

Conv2d(64, 128, kernel_size=3, padding=1) + ReLU

Conv2d(128, 256, kernel_size=3, padding=1) + ReLU + MaxPool2d(1,2)

Conv2d(256, 512, kernel_size=3, padding=1) + BatchNorm2d + ReLU

Conv2d(512, 512, kernel_size=3, padding=1) + BatchNorm2d + ReLU + MaxPool2d(1,2)

Conv2d(512, 512, kernel_size=2) + Dropout(0.5)

输出：特征图尺寸为 (batch_size, 512, 1, T)，其中 T 为序列长度。

（2）序列映射层（MapSeq）

①功能：将 CNN 输出的特征图展平为序列形式，供 RNN 处理。

②结构：

Linear(2048 → 256) + Dropout(0.5)

输入：展平后的特征向量（维度 2048）。

输出：序列向量（维度 256）。

（3）循环神经网络（RNN 部分）
①功能：建模序列依赖关系，捕捉上下文信息。
②结构：
双向 GRU 层 1：nn.GRU(256, 256, bidirectional=True)
双向 GRU 层 2：nn.GRU(512, 256, bidirectional=True)
输出：双向上下文感知的序列特征（维度 512）。
（4）输出层
①功能：预测每个时间步的字符类别。
②结构：
Linear(512 → vocab_size)：vocab_size 为字符集大小（含空白符）。
输出：形状为 (T, B, V)，其中 T 为序列长度，B 为批次大小，V 为词汇表大小。

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 80, 200]             896
              ReLU-2          [-1, 32, 80, 200]               0
         MaxPool2d-3          [-1, 32, 40, 100]               0
            Conv2d-4          [-1, 64, 40, 100]          18,496
              ReLU-5          [-1, 64, 40, 100]               0
         MaxPool2d-6           [-1, 64, 20, 50]               0
            Conv2d-7          [-1, 128, 20, 50]          73,856
              ReLU-8          [-1, 128, 20, 50]               0
            Conv2d-9          [-1, 256, 20, 50]         295,168
             ReLU-10          [-1, 256, 20, 50]               0
        MaxPool2d-11          [-1, 256, 10, 25]               0
           Conv2d-12          [-1, 512, 10, 25]       1,180,160
      BatchNorm2d-13          [-1, 512, 10, 25]           1,024
             ReLU-14          [-1, 512, 10, 25]               0
           Conv2d-15          [-1, 512, 10, 25]       2,359,808
      BatchNorm2d-16          [-1, 512, 10, 25]           1,024
             ReLU-17          [-1, 512, 10, 25]               0
        MaxPool2d-18           [-1, 512, 5, 12]               0
           Conv2d-19           [-1, 512, 4, 11]       1,049,088
          Dropout-20           [-1, 512, 4, 11]               0
          Dropout-21           [-1, 512, 4, 11]               0
          Dropout-22           [-1, 512, 4, 11]               0
           Linear-23              [-1, 11, 256]         524,544
          Dropout-24              [-1, 11, 256]               0
          Dropout-25              [-1, 11, 256]               0
          Dropout-26              [-1, 11, 256]               0
              GRU-27  [[-1, 11, 512], [-1, 11, 256]]               0
              GRU-28  [[-1, 11, 512], [-1, 11, 256]]               0
           Linear-29              [-1, 11, 63]          32,319
================================================================
```

Figure 1 CRNN 模型参数

4.1.2 YOLO

前往 Ultralytics 官网下载 yolo11n.pt 模型权重。并使用 YOLO 函数加载训练。

```
YOLO11x summary: 357 layers, 56,945,386 parameters, 0 gradients
Total parameters: 56945386
Trainable parameters: 0
```

Figure 2 YOLO11 模型参数

4.2 数据预处理

4.2.1 数据标注

（1）锚框标注

由于 YOLO 训练数据集需要锚框，所以需要在原数据集的基础上再做一次标注，即用锚框将图片内部的字符框出。

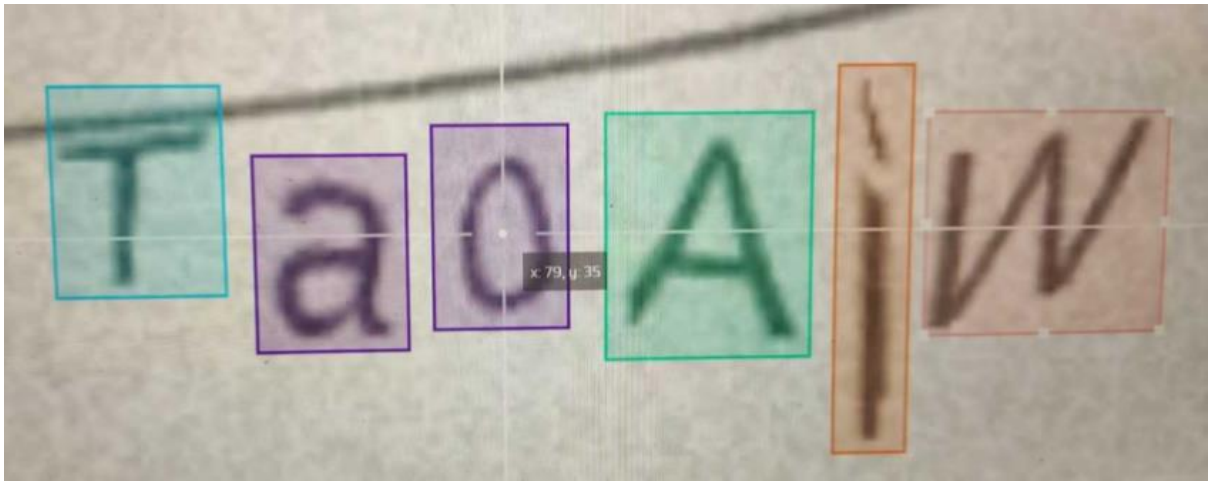


图 6 锚框标注

标注平台可以使用 Make Sense

（2）字符编码

对需要预测的每个字符进行数字编码

```
char_to_idx = {
    'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i':
8, 'j': 9,
    'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r':
17, 's': 18,
    't': 19, 'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25, 'A':
26, 'B': 27,
    'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33, 'I': 34, 'J':
35, 'K': 36,
    'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42, 'R': 43, 'S':
44, 'T': 45,
    'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z': 51, '0': 52, '1':
53, '2': 54,
    '3': 55, '4': 56, '5': 57, '6': 58, '7': 59, '8': 60, '9': 61
}
```

代码 1 字符编码

标注后的标签

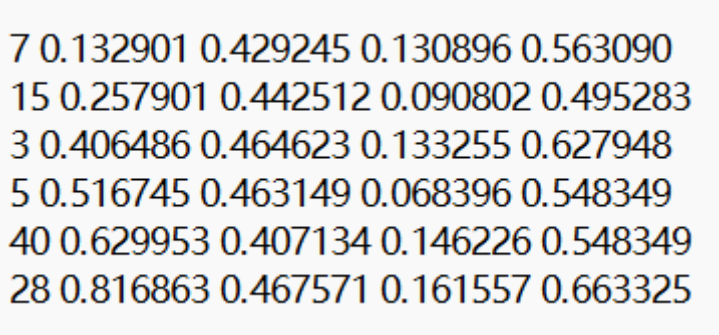


图 7 YOLO 标签

第一列的数字为字符的编码，剩余为几何位置信息。

4.2.2 数据增强

修改 yaml 文件

```
augment: True           # 默认开启增强
mosaic: 1.0             # Mosaic 增强强度
mixup: 0.2              # MixUp 概率
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
translate: 0.1
scale: 0.5
shear: 0.0
flipud: 0.0
fliplr: 0.0
```

代码 2 增强参数

经过多次尝试，对增强参数进行调整。

4.3 模型训练与测试

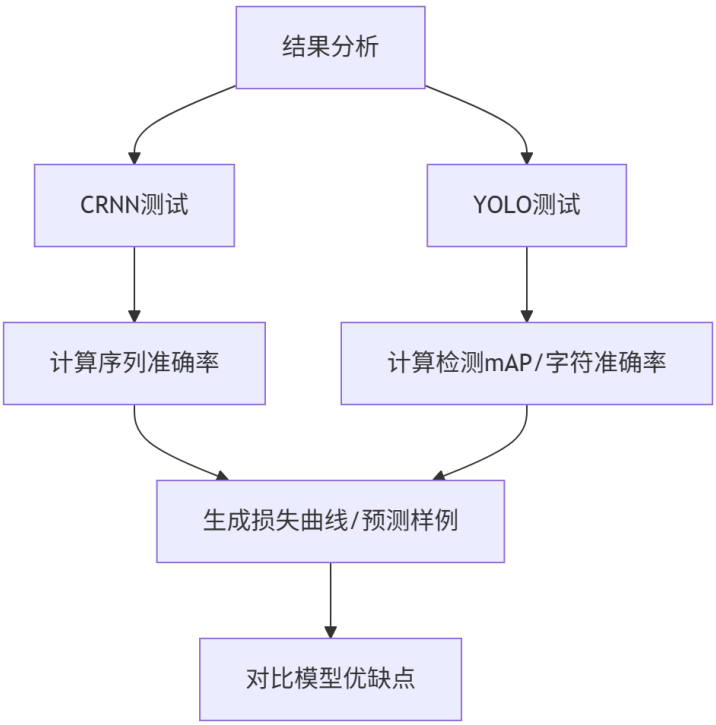


图 8 模型测试流程

4.3.1 CRNN 模型

（1）训练参数:

```
lr = 0.02
# 权重衰减
weight_decay = 1e-5
# 下降幅度
momentum = 0.7
```

代码 3 训练参数

批处理大小（BATCH_SIZE）: 8
学习率（lr）: 0.02
权重衰减（weight_decay）: 1e-5
动量（momentum）: 0.7
优化器: SGD（随机梯度下降），使用 Nesterov 动量
训练轮数（EPOCHS）: 50
工作进程数（N_WORKERS）: 0
（2）训练

```
Epoch [   38/   50] | train loss 0.0424 | val loss 1.3368
Epoch [   39/   50] | train loss 0.0358 | val loss 1.4540
Epoch [   40/   50] | train loss 0.0287 | val loss 1.8508
Epoch [   41/   50] | train loss 0.0261 | val loss 1.3000
Epoch [   42/   50] | train loss 0.0211 | val loss 1.2844
Epoch [   43/   50] | train loss 0.0173 | val loss 0.8877
Epoch [   44/   50] | train loss 0.0143 | val loss 0.8781
Epoch [   45/   50] | train loss 0.0141 | val loss 0.6175
Epoch [   46/   50] | train loss 0.0142 | val loss 0.6348
Epoch [   47/   50] | train loss 0.0102 | val loss 0.8991
Epoch [   48/   50] | train loss 0.0103 | val loss 1.6680
Epoch [   49/   50] | train loss 0.0105 | val loss 0.7850
Epoch [   50/   50] | train loss 0.0095 | val loss 0.8382
```

Figure 3CRNN 训练过程

4.3.2 YOLO11

（1）训练参数

```
model.train(data=r'valcode.yaml',
                imgsz=(80, 200),
                epochs=50,
                batch=4,
                workers=0,
                device='',
                optimizer='SGD',
                close_mosaic=10,
                resume=False,
                project='runs/train',
                name='exp',
                single_cls=False,
                cache=False,
                )
```

代码 4 YOLO 参数

批次样本数量：4
轮数：50
优化器：SGD
（2）训练

```
198/200      10.3G      1.099     0.9454      1.009       2024        224: 100%|████████| 2/2 [00:01<00:00,  1.19it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|████████| 1/1 [00:00<00:00,  2.05it/s]
               all        200       1200      0.812      0.862      0.884      0.621

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size

199/200      10.3G      1.106     0.9508      1.011       2032        224: 100%|████████| 2/2 [00:01<00:00,  1.18it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|████████| 1/1 [00:00<00:00,  2.03it/s]
               all        200       1200      0.821      0.868      0.888      0.624

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size

200/200      10.3G      1.101     0.9422      1.016       2026        224: 100%|████████| 2/2 [00:01<00:00,  1.19it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|████████| 1/1 [00:00<00:00,  2.03it/s]
               all        200       1200      0.822      0.881      0.893      0.627

200 epochs completed in 0.171 hours.
```

Figure 4 YOLO 训练过程

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	200	1200	0.834	0.903	0.903	0.636
0	16	18	0.871	0.944	0.955	0.687
1	13	13	0.369	0.769	0.486	0.36
2	25	26	0.921	1	0.991	0.693
3	22	23	0.62	0.87	0.773	0.531
4	26	27	0.919	1	0.995	0.7
5	23	23	0.916	0.944	0.907	0.621
6	20	23	0.993	0.957	0.993	0.714
7	20	20	0.735	0.95	0.926	0.616
8	22	22	0.449	0.593	0.673	0.395
9	20	24	0.551	0.75	0.585	0.368
10	20	20	1	0.891	0.979	0.688
11	16	16	0.407	0.643	0.411	0.236
12	25	26	0.832	0.923	0.845	0.602
13	21	21	0.617	0.762	0.881	0.646
14	21	21	0.842	0.905	0.954	0.665
15	26	26	0.602	0.846	0.668	0.479
16	22	24	0.394	0.583	0.553	0.372
17	19	20	0.845	0.85	0.921	0.595
18	20	20	0.944	0.849	0.951	0.689
19	14	15	0.762	0.933	0.924	0.584
20	18	18	0.929	1	0.995	0.621
21	17	18	0.94	0.869	0.906	0.631
22	22	23	1	0.831	0.955	0.658
23	12	12	1	0.7	0.995	0.662

Figure 5 训练结束

训练结束：

Class：类别标签。

Images：图像数量。

Instances：实例数量。

Box(P)：边界框精度（Precision）。

R：召回率（Recall）。

mAP50：平均精度（mAP）在 IoU 阈值为 0.5 时的值。

mAP50-95：平均精度（mAP）在 IoU 阈值从 0.5 到 0.95 时的值。

五、 实验分析

5.1 损失曲线

5.1.1 CRNN

（1）损失曲线

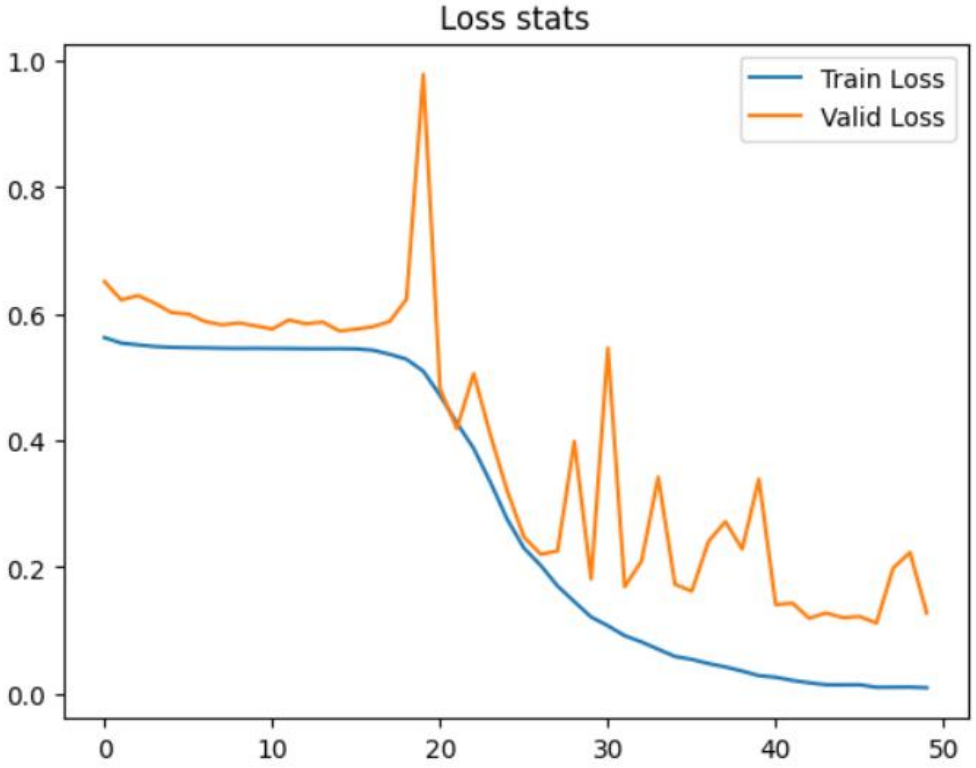


Figure 6 CRNN loss 曲线

（2）预测样例：

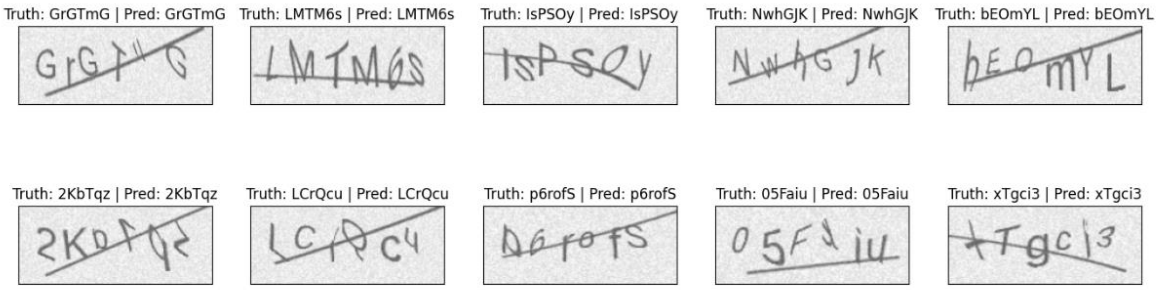


图 9 预测样例

5.1.2 YOLO

（1）损失曲线

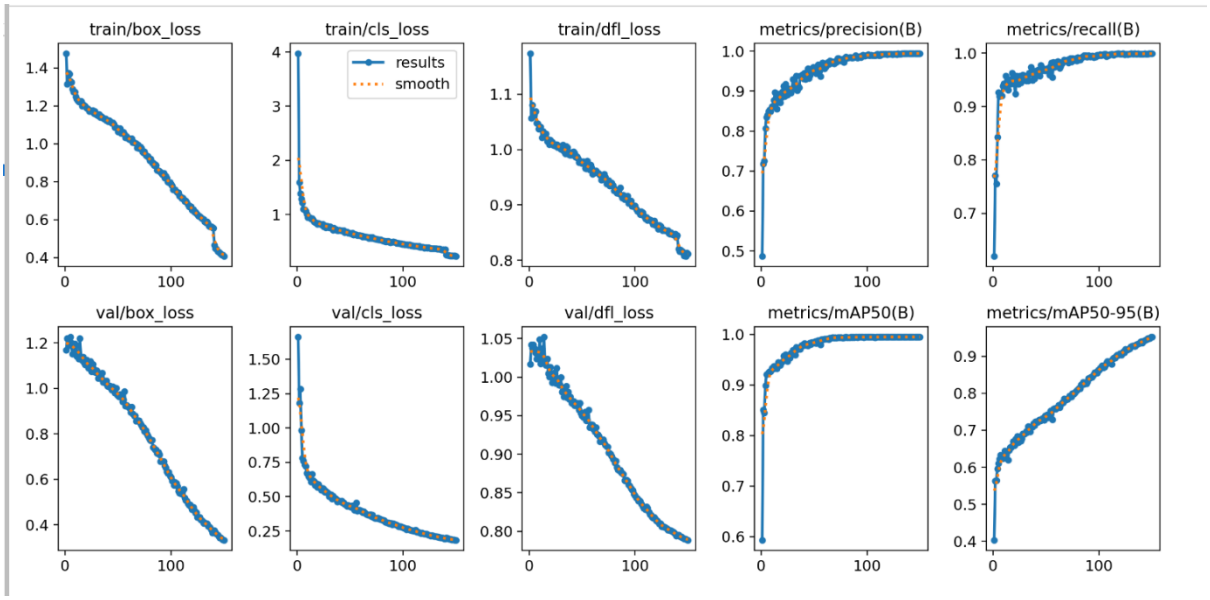


Figure 7 YOLO loss

train/box_loss：训练过程中边界框回归的损失值。

train/cls_loss：训练过程中分类任务的损失值。

train/dfl_loss：训练过程中分布聚焦损失（Distribution Focal Loss）的值。

metrics/precision(B)：验证集上的精度（Precision）。

metrics/recall(B)：验证集上的召回率（Recall）。

val/box_loss：验证集上边界框回归的损失值。

val/cls_loss：验证集上分类任务的损失值。

val/dfl_loss：验证集上分布聚焦损失的值。

metrics/mAP50(B)：验证集上的平均精度（mAP）在 IoU 阈值为 0.5 时的值。

metrics/mAP50-95(B)：验证集上的平均精度（mAP）在 IoU 阈值从 0.5 到 0.95 时的值。

（2）预测效果

```
label: 5JnIQ4, predict: JnIQ4
label: rtIYar, predict: qqIYaq
label: CFi7Gt, predict: COF7Gt
label: jCeLrM, predict: jCsLrM
label: zvk8eR, predict: zvk8qR
label: iX1Ba0, predict: Xx1Ba0
label: 9wUkfV, predict: 9wUkfV
label: eXcJyD, predict: eXcJyD
label: ZPsuzF, predict: ZPsuF
label: WRmuLl, predict: WRmuL
label: QheY8X, predict: QOaY8X
```

图 10 在测试集上测试

5.2 效果对比

（1）CRNN：

在训练集上的表现：

```
✅ 加载已有模型权重：ocr.pth
Accuracy: 0.9667
```

图 11

在验证集上的表现：

```
预测文本：9Ppcf4，真实文本：9Ppcf4
预测文本：jPHZbG，真实文本：jOxZbG
预测文本：lFE4Qg，真实文本：IFEA0g
预测文本：URzVGG，真实文本：LRz1GG
预测文本：70jDRM，真实文本：70jDRM
预测文本：YHdhFX，真实文本：YHdhFX
预测文本：TuaV，真实文本：qqTuaV
预测文本：xT83Cu，真实文本：x733Cu
Accuracy: 0.2500
```

图 12

在测试集上的表现：

```
准确率：23.75%
```

图 13

（2）YOLO：

```
准确率：72.15%
```

准确率相较于 CRNN 有明显的提升。

六、 问题和解决方案

6.1 数据集较少

由于 1000 张图片作为数据集数量较少，所以需要在 YOLO 模型训练时使用数据增强。

而 CRNN 模型由于数据增强效果在验证码类型的数据上并不显著，所以采用生成类似数据的方式扩充数据集。

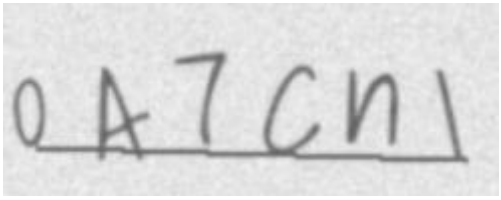


图 14 生成的数据

但是在生成数据后 CRNN 的表现仍然差于 YOLO11。

七、 实验总结

7.1 实验结果总结

（1）CRNN 模型：

在训练集上表现优异（准确率 96.67%），但在验证集和测试集上泛化能力不足（测试集准确率仅 23.75%）。分析原因：

序列依赖性：CRNN 依赖字符顺序，而验证码中字符可能随机排列，导致预测错误。

数据增强局限性：传统图像增强（如旋转、噪声）难以模拟验证码的复杂干扰（如曲线干扰线），模型易过拟合。

（2）YOLO 模型：

测试集准确率达 72.15%，显著优于 CRNN。其优势体现在：

目标检测特性：直接定位并分类单个字符，不受字符顺序影响。

鲁棒的数据增强：Mosaic 增强和 MixUp 策略有效模拟验证码的复杂场景，提升模型抗干扰能力。

7.2 改进方向

尽管 YOLO 表现更优，仍有以下优化空间：

小目标检测优化：针对验证码中的小字符，可引入更高分辨率的特征图（如 P2 层）或注意力机制（如 CBAM）。

后处理优化：结合字符位置信息对 YOLO 输出结果排序，解决无序字符的序列重组问题。

混合模型尝试：将 YOLO 的检测能力与 CRNN 的序列建模结合，例如先用 YOLO 定位字符，再通过 CRNN 细化识别。

参考文献：

[1] crnn and ctc. (2021, July 6). Zhihu. https://zhuanlan.zhihu.com/p/43534801
[2] libo, coder. (2020). *『OCR_Recognition』CRNN-CSDN 博客*. Csdn.net. https://blog.csdn.net/libo1004/article/details/111595054
[3] https://zhuanlan.zhihu.com/p/7247937963
[4] https://blog.csdn.net/xcg340123/article/details/136484474