《智算视界》设计说明

2025 年 4 月

目录

摘要

本课程设计旨在设计并研发一款手写矩阵识别与分步可视化计算过程展示系统，解决学生在学习数学课程中面临的抽象公式和复杂计算步骤难以理解的问题。结合人工智能与可视化动画技术，系统通过引入 Mamba 主干网络 提升对手写矩阵内容的精准识别，利用 YOLO 模型 完成数字提取与矩阵还原，同时结合 CoordAttention 坐标注意力机制，增强对字符空间位置的精确把握。计算过程通过 Manim 动画引擎 逐帧演示，生动展现每一个数学计算步骤，帮助学生直观理解数学运算的内在逻辑。通过动态生成不同运算任务的动画流程，系统具有高度的模块化和灵活性，能够根据学生需求实时生成不同的数学运算示例。通过创新的人机交互方式，将数学计算过程“动起来”，帮助学生在深度理解的基础上跨越知识与理解之间的鸿沟，最终实现“看懂每一步”的学习体验。系统的可扩展性和互动性使其成为教育教学中的直观、友好且高效的辅助工具。

1 引言

1.1 动机

在高等教育日益强调能力导向和实践导向的今天，学生不仅需要积累知识，更面临着如何理解与应用这些知识的挑战。尤其在数学类课程中，许多学生往往被复杂的公式和计算步骤所困，无法真正理解其中的推导过程。特别是线性代数、矩阵计算等课程，学生不仅缺乏对公式的直观理解，且在计算时常常出现错误，无法有效建立起对知识的系统认知。这一问题在当前的教育环境中尤为突出，尤其是在一些高年级课程的学习过程中，学生的知识理解仍停留在表面，缺乏深入的思考与实践。随着信息技术的发展，市面上虽然有一些工具如微软数学、MathDF 等，提供了一定的计算支持，但这些工具大多偏重于给出最终结果，忽视了学生对计算过程的需求。更为重要的是，这些工具的个性化和互动性较差，无法针对不同学生的学习进度和理解水平提供定制化的支持。

1.2 要解决的问题

数据集生成 在数学学习与计算领域，尤其是矩阵运算的教学中，现有的数据集多集中于图像识别、文本分类等领域，而专门用于数学矩阵识别的数据集相对较少。这使得我们在构建训练集时面临了一定的困难。尤其对于手写矩阵的识别，数据集的不足导致了对不同书写风格和矩阵规模的覆盖不够全面。

图像识别的准确度 现有的目标检测算法，YOLO（You Only Look Once）[1]，已被广泛应用于物体检测和图像分类等任务，并在实时性和准确度方面表现出色。然而，手写矩阵的图像识别面临着许多挑战，例如字体变化、噪声干扰、图像模糊等问题，这些都会影响识别的精度。

数学算式定位 在完成矩阵图像的识别后，接下来的一大挑战是如何将识别到的元素与数学算式对应的位置进行精确定位。不同的矩阵计算涉及到不同的数学运算符，如加法、乘法、行列式等，这些符号在数学算式中的位置和作用需要被准确地识别和定位。

代码框架可拓展性 随着技术的不断发展和需求的变化，系统的可维护性和可拓展性显得尤为重要。在系统的设计阶段，采用模块化的开发方式，确保各个功能模块之间的依赖最小化，从而便于后期对功能模块的修改与升级。以及后续的前后端界面如何快速的更新和实时的协作扩展功能。

1.3 本文的工作

为了解决上述问题，本项目结合人工智能与可视化动画技术，提出了一种创新的人机交互方式。本系统通过集成 YOLO-Mamba[2]图像识别技术和 Manim 动画引擎，能够精准地识别学生上传的手写矩阵，并通过动态动画的形式展示整个计算过程。与传统的数学教学工具不同，本系统不仅注重计算结果的展示，更注重过程的展示，通过清晰的可视化推导帮助学生理解每一个运算步骤的逻辑与原理。

图像识别 本项目采用 YOLO-Mamba 目标检测算法，通过优化的训练数据集，提升了对手写矩阵的准确识别能力。YOLO 算法通过多层卷积神经网络的学习，能够有效地识别出矩阵中的各个元素，克服了手写字体变化、噪声干扰及图像模糊等问题，确保了高精度的识别结果。

动画流程的自动化生成 系统能够根据用户输入的矩阵大小、类型以及所需的数学运算，动态生成相应的计算过程动画，而无需人工干预。这一机制基于 Manim 动画引擎的模块化子组件和统一的数据接口，使得系统能够实现高度的灵活性和可扩展性。无论是不同规模的矩阵运算，还是不同行列式、加法、乘法等运算，系统均能自动生成相应的动画效果，极大提高了用户体验与交互性。

可维护性和可拓展性 为了确保系统的可维护性和可拓展性，本项目采用了模块化开发方式，确保各个功能模块的独立性和可扩展性。通过统一的数据接口和模板化结构，系统能够轻松扩展新的数学运算功能，并且在后期能够快速更新与维护。特别是在前后端协作方面，系统的设计确保了新的功能可以迅速上线，并支持多任务、多场景的应用，使得系统能够应对不断变化的教育需求。本文的所有工作已经上传至 https://github.com/rainbowyuyu/animate_cal，所有工作均可通过 README 的说明进行复现，通过请求即可进行拓展和修复维护。可视化平台地址为 https://animatecal-aesrxwe852bslylhgvrfxx.streamlit.app/，通过访问 GitHub 流实时更新前端。

2 相关工作

2.1 手写数字识别

手写数字识别技术已经取得了显著进展，尤其在深度学习的发展过程中，基于卷积神经网络（CNN）[3]的方法逐渐成为主流。CNN 通过多层卷积和池化操作能够高效地提取图像中的特征，特别是在处理多样化字符和复杂背景时，比传

统的光学字符识别（OCR）[4]方法如边缘检测、颜色分析和模板匹配等更具优势。虽然在标准数据集上取得了良好的效果，但往往需要较长的训练时间和计算资源。因此，采用更加高效的 CNN 结构，尤其是在处理带噪音和复杂背景的手写数字时，仍然是一个值得探索的方向。

2.2 角点检测算法

角点检测是计算机视觉中的关键任务，广泛应用于图像匹配、运动跟踪等领域。Harris 角点检测算法[5]通过自相关矩阵评估局部窗口的灰度变化，判断图像中是否存在角点。Shi-Tomasi 算法[6]作为 Harris 算法的改进，采用了局部亮度梯度信息和结构矩阵特征值来进行角点检测，能够更好地抵抗噪声并提高鲁棒性。尽管有着更强的适应性，但其计算复杂度较高，且需要大规模的标注数据进行训练。

2.3 YOLO 神经网络

YOLO 算法是一种基于卷积神经网络（CNN）的实时目标检测方法。YOLO 的核心创新在于将目标检测任务转化为一个回归问题，通过对整个图像的单次前向传播，直接输出目标的类别及其边界框坐标。与传统的目标检测方法不同，YOLO 在空间分辨率较低的情况下，依然能够有效地捕捉图像中的小目标，并减少背景误识别的几率。但对于手写算式这类需要捕捉图像的细节和上下文信息，和添加注意力识别这类工作仍需改进。

2.4 Mamba 主干网络

Mamba 主干网络是近年来提出的一种视觉任务中的卷积神经网络架构，旨在改进传统卷积网络在复杂视觉任务中的表现。与其他主干网络相比，Mamba 网络通过引入动态卷积和自适应特征融合机制，能够更有效地捕捉图像的细节和上下文信息。

3 数据收集与标注

3.1 原始数据集

原始数据集“Handwritten math symbols dataset”共有 82 个类，包含各自运算符号和数字字母，每个类中有大约 10000 张 45×45 的 jpg 格式的黑白图像。数

据集网址为 https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols/data。

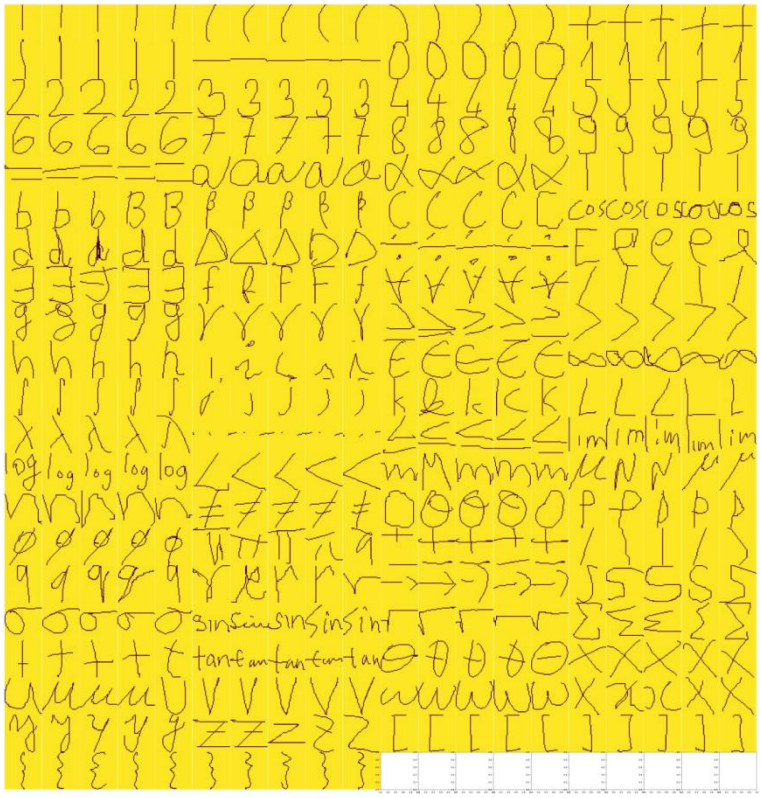


图 3.1 原始数据集样例图

3.2 数据集生成

数据集生成采用数字图像处理的添加 alpha 通道、图像横纵变换、添加噪点等操作，进行了多版本的迭代和尝试，最后实现了自动生成数据集和自动标注功能。

数据集生成器的版本迭代也是模型版本迭代的重要依据，后面介绍模型时的版本性能迭代的依据就是数据集的版本。

数据集的自动生成和自动标注，解决了人手写数据集和 labelimg 标注的庞大工作量。

表 3.1 数据集版本汇总

版本	功能
v0	将数字符号随机排列
v1	添加 alpha 通道
v2	生成矩阵元素
v3	生成矩阵

v4	生成稳定的手写矩阵

（1）v0

读取所有文件建成一个列表，随机选取一个元素随机大小摆放在随机位置。标注生成时，对标注的 width 和 height 做随机值的改变。

v0 产生的问题：上层的元素覆盖下层，标签中的图片不全，产生非常严重的过拟合。

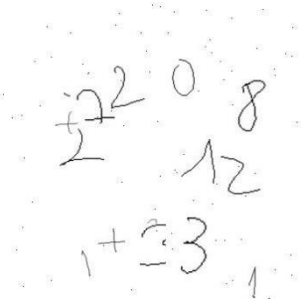


图 3.2 v0 生成的图片样例

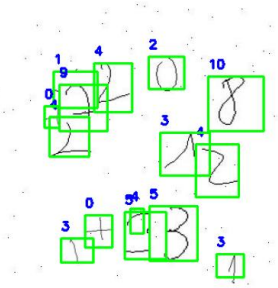


图 3.3 v0 生成的标注样例

注：图像中标注的为图像的类别，其 classes 映射关系为：[{0:+},{1:-},{2:0},{3:1},{4:2},{5:3},{6:4},{7:5},{8:6},{9:7},{10:8},{11:9},{12:=},{13:[},{14:]},{15:times}]，3.1.2 部分中所有图中所有标注都依据此映射关系呈现。

（2）v1

针对 v0 版本遇到的问题，做了如下调整：

- 减少了元素的数量，对原始数据集添加了 alpha 通道，变为了 png 的透明灰度图像。这样就不会存在上层图像白色背景覆盖掉下层图像的情况。

- 生成的图像不再是固定的正方形图像，而是随机适当改变总图片的大小，减小过拟合。

v1 产生的问题：多个元素互相叠加覆盖，标注的两个标签中的元素叠加在一起无法区分，产生严重的过拟合。

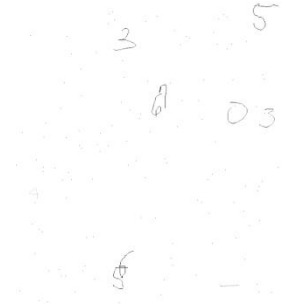


图 3.4 v1 生成的图片样例

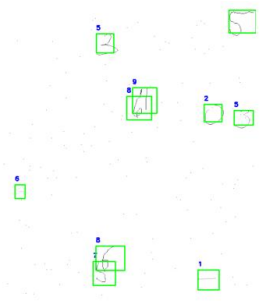


图 3.5 v1 生成的标注样例

（3）v2

针对 v1 版本遇到的问题，做了如下调整：

- 更改随机点生成逻辑，不直接从列表中添加图片随机摆放，而是先随机生成 n 个点，保证这 n 个点的间隔，再定位这 n 个点为图像的左上点，避免了图像重复的情况。

- 重写了 alpha 通道的生成方法，减小了去除白色背景的阈值，防止元素特征过少。

v2 已经能够稳定生成基本的多元素数据集，继续生成矩阵和矩阵中的元素，使得特征更多样化，具有泛化，通过生成矩阵元素后自适应拼接实现。

生成矩阵元素：根据实际情况，生成 1-3 个数字，当位数>2 时，第一位有概率为负号，适当随机数字的大小和间隔排列。

生成矩阵：编写了自适应元素多少的图片大小调整算法，以及自适应行列数的矩阵元素摆放算法。

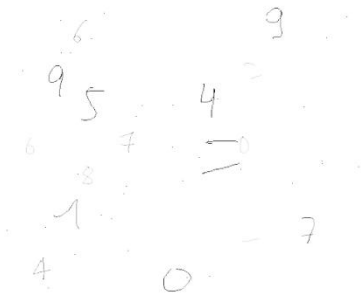


图 3.6 v2 生成的 alpha 图样例

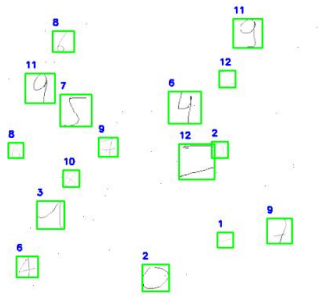


图 3.7 v2 生成的 alpha 标注样例

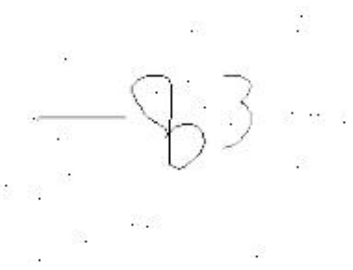


图 3.8 v2 生成的矩阵元素图样例

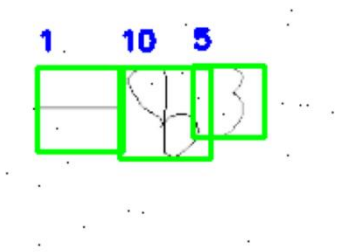


图 3.9 v2 生成的矩阵元素标注样例

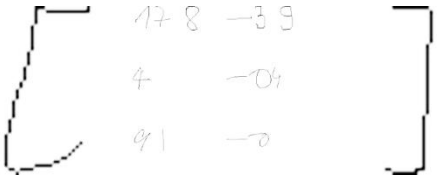


图 3.10 v2 生成的矩阵图样例

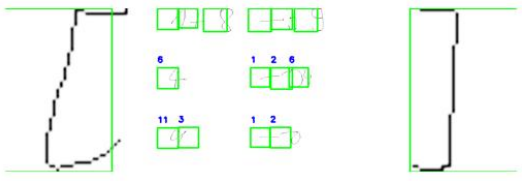


图 3.11 v2 生成的矩阵标注样例

（4）v3

针对 v2 版本遇到的问题，做了如下调整：

- 修复 v2 版本中的标签错位问题。

- 调整矩阵的左右括号的自适应纵向拉伸关系，上下左右的随机移动和，以模仿人手写的矩阵。

- 添加了矩阵中元素的自适应居中算法，为后续聚类分割做准备，更模仿了人手写矩阵的特点。

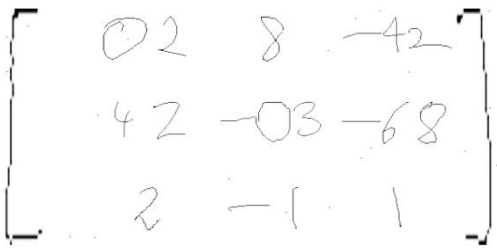


图 3.12 v3 生成的矩阵图样例

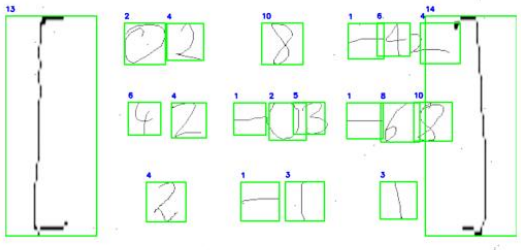


图 3.13 v3 生成的矩阵标注样例

（5）v4

针对 v3 版本遇到的问题，做了如下调整：

- 调整矩阵中元素的粗细，调整 alpha 元素的生成逻辑，元素添加到矩阵时再做添加 alpha 通道的处理。

- 调整噪点的生成逻辑，噪点现在由中心向四周 alpha 值逐渐下降，模仿实际图像预处理的噪点。

- 优化了矩阵自适应位置的居中、行列位置、间隔等特点，现在矩阵以及非常接近手写矩阵。

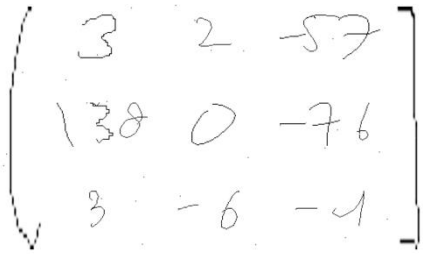


图 3.14 v4 生成的矩阵图样例

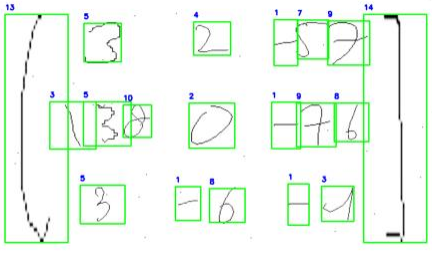


图 3.15 v4 生成的矩阵标注样例

通过多版本的迭代对矩阵生成算法的优化和调整，获得最终稳定版的矩阵自动生成和标注数据集，进入模型训练，共 300000 张，训练集和测试集比例为 9：

1，数据集的样例如图 3.16 所示。

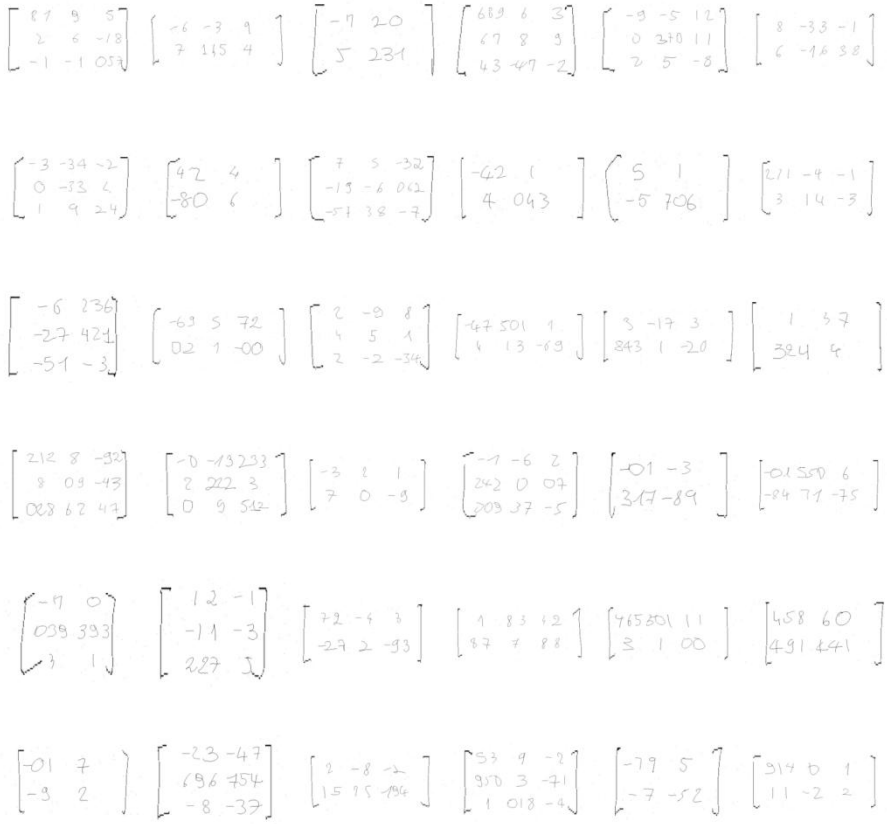


图 3.16 生成数据集样例图

最终稳定版的算法流程图如图 3.17 所示，具体代码见附录 I 算法 1。

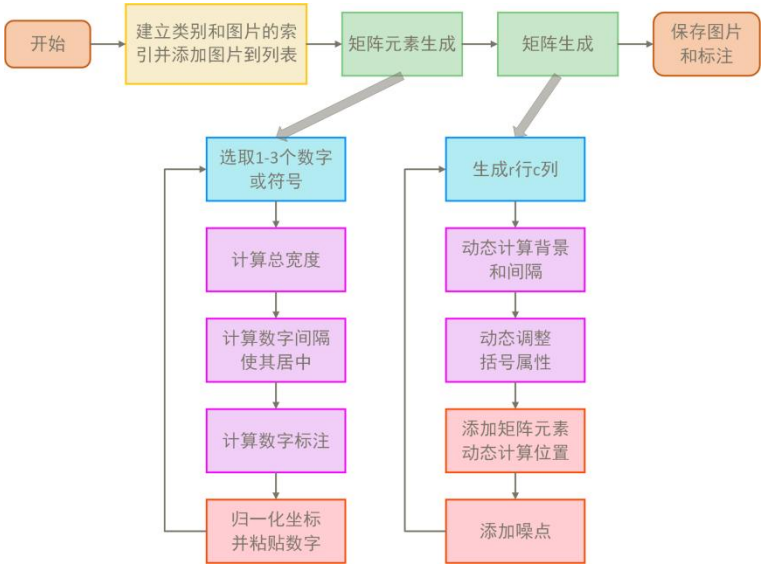


图 3.17 生成数据集算法流程图

4 关键方法

4.1 数学算式检测

4.1.1 数学算式模型训练

在网络结构与模型训练方面，本项目选用了轻量化的 YOLOv11n 作为基础目标检测框架，并针对手写算式图像的空间分布特性，进行了结构级和机制级的双重优化。由于融入了 YOLO，Mamba 和 CoordAttention 模块，我在下文把网络叫做 YMCANet，网络架构整体分为 Backbone、Neck 和 Head 三大模块。

Backbone 层在原有 C2F 结构的基础上，加入了更具表达能力的 C3K2 多尺度卷积模块，显著提升了对复杂边缘与笔迹特征的捕捉能力。为进一步增强网络的性能，引入了 Mamba 主干网络，提升了模型的整体表达能力和计算效率。此外，网络中还添加了 VSS、VCM 和 XSS，这些模块有效增强了多尺度信息的融合能力与特征提取精度。

在注意力机制方面，对原有的 PSA 模块进行了优化，替换为 CoordAttention 坐标注意力机制[7]，使得网络在识别矩阵时，不仅能够关注内容信息，还能更加精准地把握字符在图像中的空间位置，从而提升了识别精度和鲁棒性。

（1）VSS 视觉状态空间模块

VSS 模块通过引入视觉状态空间，能够有效地对图像中的视觉元素进行建模。在传统的视觉识别任务中，图像中的信息通常是静态的，而在 VSS 中，每个视觉元素都被视为一个动态状态，并且这些状态在图像处理过程中能够发生变化。该模块能够捕捉到视觉数据中不同层次的动态信息，例如，物体的变化、背景的变化，以及视觉元素之间的相互关系。

（2）VCM 卷积特征融合模块

VCM 模块主要用于融合来自不同卷积层的特征信息，以增强网络对复杂图像特征的理解能力。它通过将多个卷积层的输出特征进行加权融合，确保网络能够有效地整合不同层次的信息，提升特征的多样性和深度表达能力。VCM 使得网络能够更好地处理具有不同尺度和分辨率的图像信息，从而提高了特征提取的精度，尤其在处理复杂场景或图像时表现优秀。

（3）XSS 扩展感受野模块

XSS 模块通过扩展卷积核的感受野，能够捕捉更加广泛的上下文信息。该模块利用多层卷积操作，有效扩展了网络对图像中远距离像素的感知能力，使得模型在处理大范围区域内的特征时更加精准。XSS 模块对于图像中的长距离依

赖关系和复杂背景具有较强的捕捉能力，提升了网络在多尺度特征融合方面的表现，尤其在图像分辨率较高或结构复杂的场景中具有显著优势。

（4）CoordAttention 坐标注意力机制

CoordAttention 是一种基于空间坐标的注意力机制，与传统的通道注意力机制不同，它不仅关注图像中的内容信息，还通过引入坐标信息来精准把握字符在图像中的空间位置。通过对坐标位置的加权，CoordAttention 可以使得网络更加聚焦于关键区域，提升对空间结构和布局的理解能力。在识别矩阵时，CoordAttention 能够准确地定位字符的具体位置和形状，从而提升识别精度和鲁棒性，特别是在复杂背景或字符扭曲的情况下。

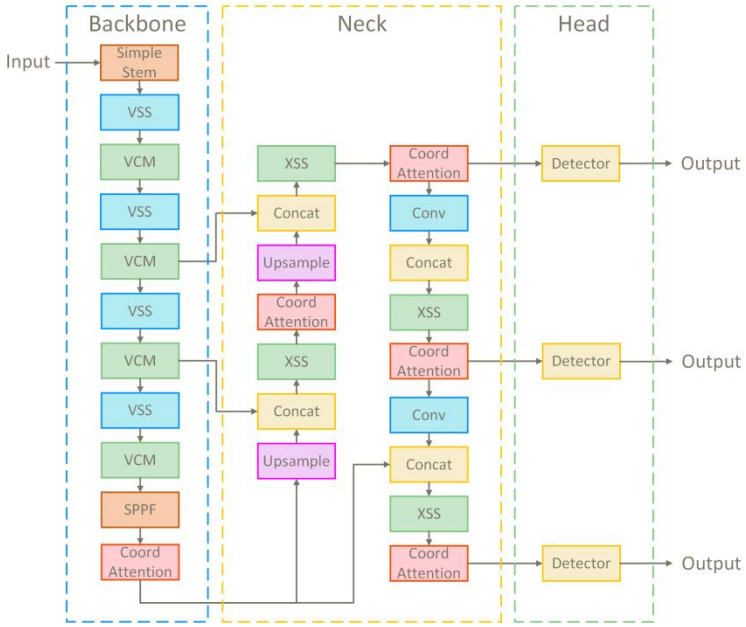


图 4.1 YMCANet 网络模型图

CoordAttention 坐标注意力机制可以捕捉特定位置的空间关系，并在注意力计算中加以利用，其算法流程图如图 3.6 所示。

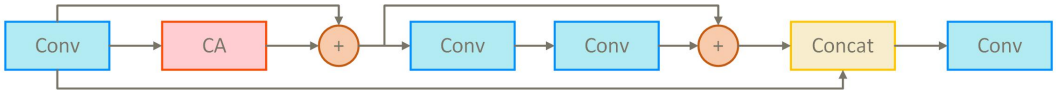
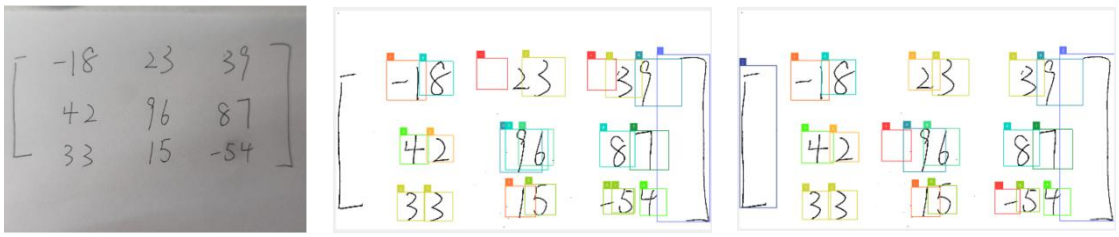


图 4.2 CoordAttention 模块算法流程图

算法计算公式：

$$\text{output} = \text{Conv1} \times 1(\text{concat}(a, + \text{Cooratt}(b) + \text{FFN}(b))) \quad (1)$$



原图　　PSA 模块　　CA 模块

图 4.3 注意力模块效果对比

训练阶段使用了自研的手写矩阵数据集（共 30 万张图像，包含多种排列与干扰形式），在 10 轮训练内模型即可达到 maP50 0.95 以上，展现出良好的收敛速度与泛化能力。此外，训练过程在多代模型之间进行预训练迁移，结合精细调参与层级可视化，有效提升了模型在复杂书写风格下的识别稳定性，为后续的分步可视化运算提供了坚实的识别基础。

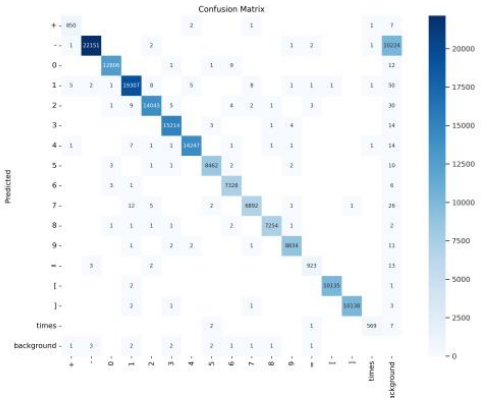


图 4.4 v4 生成的矩阵图样例

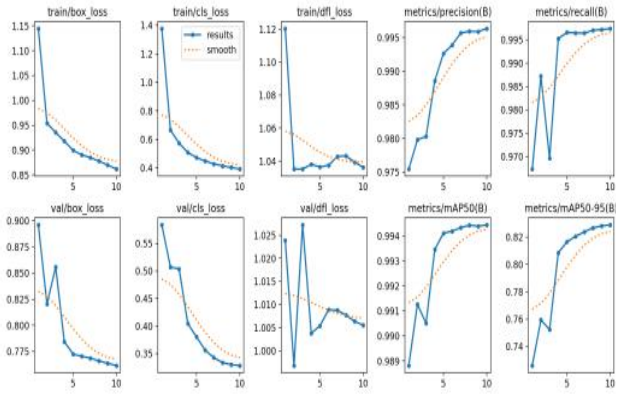


图 4.5 v4 生成的矩阵标注样例

4.1.2 算式图像预处理

由于生成的数据集是二值化的，所以需要对输入的拍摄的图像进行图像预处理算法，以及进行适当的缩放预处理，具体算法流程图为图 3.10。

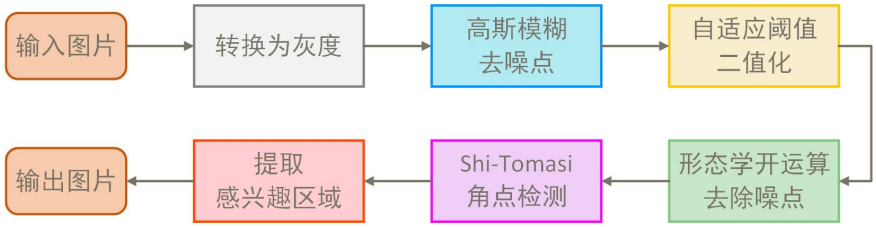


图 4.6 图像预处理算法流程图

（1）转换为灰度图像和高斯模糊去噪：

应用 cv2.GaussianBlur 对图像进行平滑处理。通过卷积操作，使用一个 5×5

高斯核对图像进行模糊，根据核大小自动计算标准差。能够平滑图像，去除细小的噪声，使得后续的阈值化和形态学处理更加准确和可靠，具体运算公式如下：

$$I_{\text{blurred}}(x,y)=\sum_{i=-k}^{k}\sum_{j=-k}^{k}I(x+i,y+j)K(i,j) \tag{2}$$

（2）自适应阈值处理：

将图像二值化，只保留黑色和白色，便于区分前景和背景。使用 cv2.adaptiveThreshold 方法，根据每个像素周围区域的亮度自动调整阈值，从而有效应对光照不均的情况。使用了 cv2.ADAPTIVE_THRESH_GAUSSIAN_C 方法，邻域内像素的加权平均值作为局部阈值，从而适应不同光照条件下的图像，自适应阈值具体运算公式如下：

$$T(x,y)=\frac{1}{N}\sum_{i=-\left\lfloor\frac{N}{2}\right\rfloor}^{\left\lfloor\frac{N}{2}\right\rfloor}\sum_{j=-\left\lfloor\frac{N}{2}\right\rfloor}^{\left\lfloor\frac{N}{2}\right\rfloor}I(x+i,y+j) \tag{3}$$

（3）形态学开运算去除噪点：

清理图像中的小噪点，增强目标区域的连通性。使用了开运算，先腐蚀后膨胀，来去除小的白色噪点。使用 cv2.morphologyEx 函数与 cv2.MORPH_RECT 结构元素矩形形状，对图像进行处理，增强后续角点检测的效果，具体运算公式如下。

腐蚀操作：

$$E(x,y)=\min_{(i,j)\in S}I\,(x+i,y+j) \tag{4}$$

膨胀操作：

$$D(x,y)=\max_{(i,j)\in S}E\,(x+i,y+j) \tag{5}$$

（4）Shi-Tomasi 角点检测：

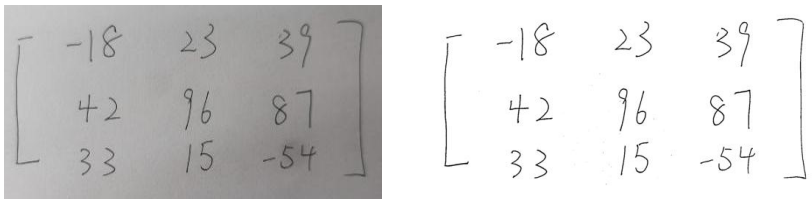
使用了 Shi-Tomasi 角点检测算法 cv2.goodFeaturesToTrack，检测图像中变化剧烈的地方，相较于常用的 Harris 和 Fast 聚类提高了精度，并且添加了最好角点的选取。限制最多提取 100 个角点，并选取质量较高的角点。角点检测提供了目标区域的位置信息，帮助后续步骤更精确地提取感兴趣区域，每个像素点的邻域的梯度协方差矩阵公式如下：

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \tag{6}$$

（5）边界范围计算和感兴趣区域提取与转换：

通过提取的角点坐标，计算出感兴趣区域的最小和最大边界，并进行边缘偏移进一步确保这些边界不会超出图像的实际范围。提取切片，并返回彩色图像。

预处理最终运行结果如图 4.7。



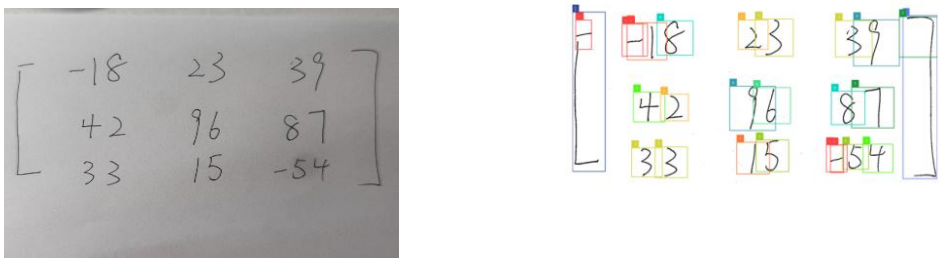
原图　　预处理结果

图 4.7 预处理结果图

4.1.3 算式内容检测

在算式内容检测模块中，系统调用了 ultralytics 提供的 YOLOv11 模型，并借助 supervision 库对检测结果进行统一监管和处理，主要返回每个识别框的中心坐标、宽高以及类别标注信息。为了提高识别准确率并避免重复识别问题，系统引入了基于 IoU（交并比）的去重蒙版机制，通过计算两个检测框的重叠比例，筛除置信度较低的冗余检测结果，从而有效降低过拟合风险。此外，系统还生成了每个元素的蒙版图，为后续的 DBSCAN 聚类分割与矩阵结构重建提供了空间基础。这一模块不仅保证了手写字符的准确定位，也为整个矩阵的结构识别与可视化计算奠定了坚实的数据支撑。

系统还添加了一个返回值返回所有标注的蒙版，为后续聚类分割做准备，处理结果如图 4.8。



原图　　预处理结果

图 4.8 yolo 检测结果图

4.1.4 数学算式数字提取

接受到 4.1.3 检测的返回值后，对矩阵进行聚类分割，知道其行列数。并且绘制分割线，为后续确定每个单独的数字符号在哪个区域做准备，最终返回横向和纵向分割线的列表，算法流程图如图 4.9.

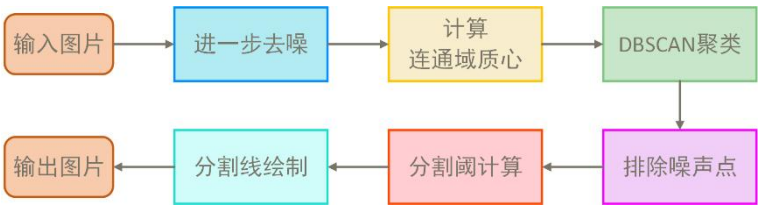
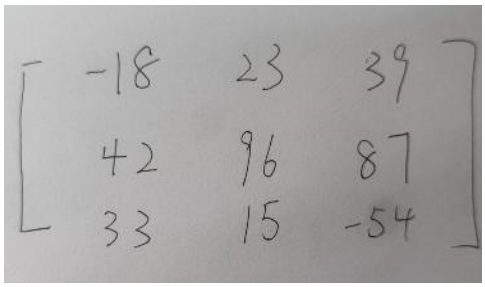


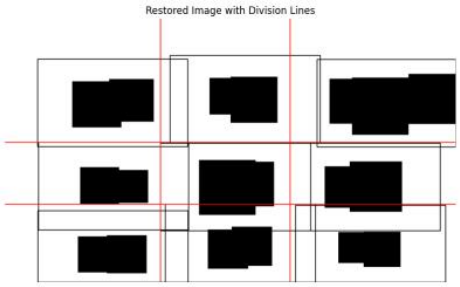
图 4.9 聚类分割算法流程图

使用 cv2.findContours 查找图像中的连通区域，通过 cv2.moments 图像矩来计算连通区域的质心，给定一个点 p 和半径ϵ，如果有足够的邻居点在此半径内，则认为这些点属于同一个簇， 根据质心的数量开根号初步计算区域网格的行列数，对于每个质心(cX, cY)，计算其所属的矩形区域，采用聚类分割域的交点来绘制多条分割线，再使用去重的操作进行分割线的筛选。

分割线验证检查某条分割线两侧是否都有质心，如果不满足则排除掉这条分割线，来确保每条分割线确实是矩阵的行列分割线。最终得到两个列表，横向和纵向的分割线，返回给数字提取模块，结果如图 4.10。



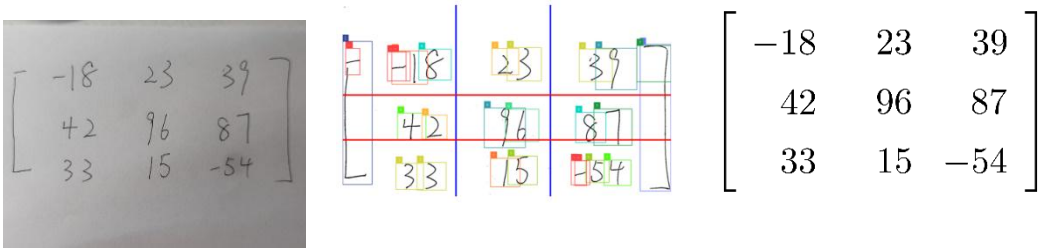
原图



预处理结果

图 4.10 聚类分割算法结果图

先通过归一化操作知道准确的每个元素之间的相对位置，为了在放入每行每列对于的元素时保证其原有的位置关系不变，先对每个元素的 x_center 进行排序，加入一个列表。遍历整个列表，判断元素 x_center 和 y_center 与图 4.10 聚类分割算法中返回的分割线的相对位置关系将元素放入对于的位置，将字符串拼接后做类型转换返回一个 numpy 的 array 格式便于后续的 Latex 呈现，结果如 4.11。



原图　　分割线结果　　Latex 矩阵展示

图 4.11 数字提取结果图

4.2 分步解法动画模块

在动画模块构建方面，系统基于 Python 的数学可视化引擎 Manim 进行底层图形渲染，并结合 ffmpeg 完成逐帧视频输出，实现了矩阵运算过程的动态可视化演示。

通过继承 Manim 的基础图形对象和文本类，构建了自定义类 SquTex，该类封装了每个矩阵元素的方块和文字组合，实现对元素的绝对位置控制、格式样式调整及动画逐个呈现操作。随后基于 SquTex 设计了 MatrixCal 类，实现二维矩阵对象的逻辑表达与排布控制，支持对每个元素的高精度定位与括号标注。在运算层面，针对行列式的计算，开发了 MatrixDet 类，支持从主对角线与副对角线分别提取乘法路径并计算中间值，调用封装的动画方法以斜向形式分步展示每组乘法与加法过程，突出显示关键路径与结果推导逻辑。而对于矩阵加法与乘法，MatrixMath 类引入双输入结构，将两个原始矩阵作为前后操作对象，并逐元素实现加法对齐与乘法的逐项展开，通过构造进度向量组与计算注释文本，实现加法框选与乘法加权累积的动画表现，提升了计算过程的透明度与可解释性。整个动画系统采用模块化设计，各类继承清晰、功能单元独立，确保可扩展性与复用性，是将抽象数学计算逻辑以专业、动态的方式进行呈现的核心支撑组件，具体类对象的继承关系如图 4.12 所示。

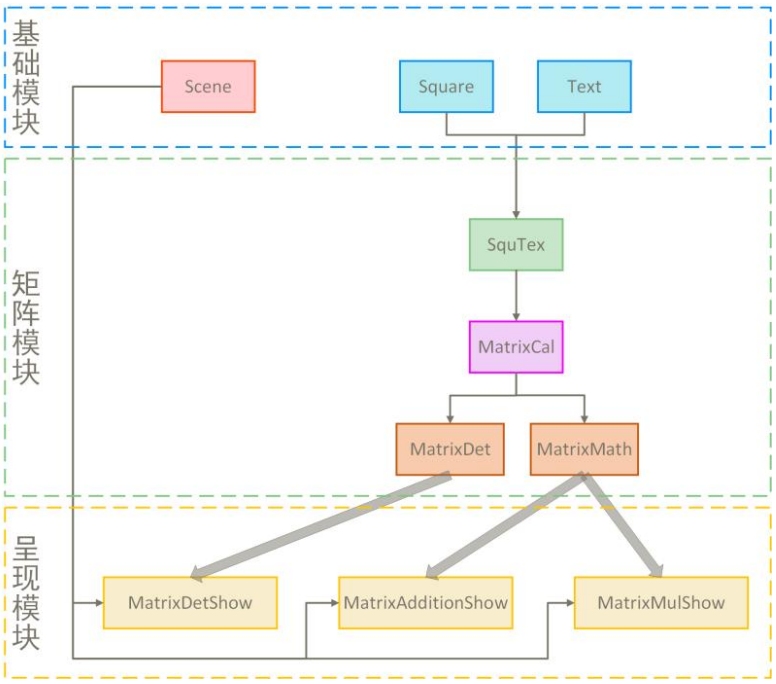


图 4.12 动画模块类对象继承关系图

4.2.1 分步解法逐帧渲染方法

Manim 引擎通过继承 Scene 或 ThreeDScene 提供了完整的矢量动画场景管理系统。本项目以 construct() 函数作为动画主入口，在该函数中逐步调用图形对象与动画控制指令，实现动态运算演示。所有渲染结果均以矢量图形（SVG）方式进行帧级生成，最终通过调用 ffmpeg 进行帧合成与编码压缩，输出为高质量的 MP4 视频。此机制具备良好的分辨率扩展性和动画流畅度，满足高复杂度数学表达式的可视化需求。

图 4.13 基础模块示例图

4.2.2 每个动画元素的控制方案

通过继承图二基础模块中所展示的 Square 和 Text 类，定义一个元素为一

个正方形和文字的组，创建构造时，输入一个列表或字符串为可变参数，创建一个方块文字组合的列表，具体构造函数如代码 1 所示。

代码 1 SquTex 构造函数

```
Input  : tex: string or list, font: string, kwargs: additional arguments
         for Square
Output: Constructed object with arrangement of squares and text
self.tex ← tex;
Call parent constructor;
for i = 0 to len(tex) - 1 do
    Create a VGroup object;
    Add a Square object to VGroup with kwargs;
    Add a Text object to VGroup with the current element of tex and
     specified font;
    Add VGroup object to current object;
end
Arrange objects with zero spacing;
```

SquTex 是动画系统中的基础图形组合类，通过继承 Manim 的 VGroup 对象，将每一个数学元素建模为“方形边框 + 文本”的复合结构，具备高精度的位置管理与样式控制能力。在构造阶段，可接受字符串列表输入，自动生成一组排布有序的图形单元。该类集成了位置排序、符号添加、样式修改、单元动画分组等方法，支持对矩阵中任意元素进行定位、突出、高亮与样式切换。通过封装如 animate_one_by_one()、add_bracket() 等动画接口，实现了对矩阵演示过程的逐帧控制与多样化视觉效果。具体效果如图 4.14 所示。

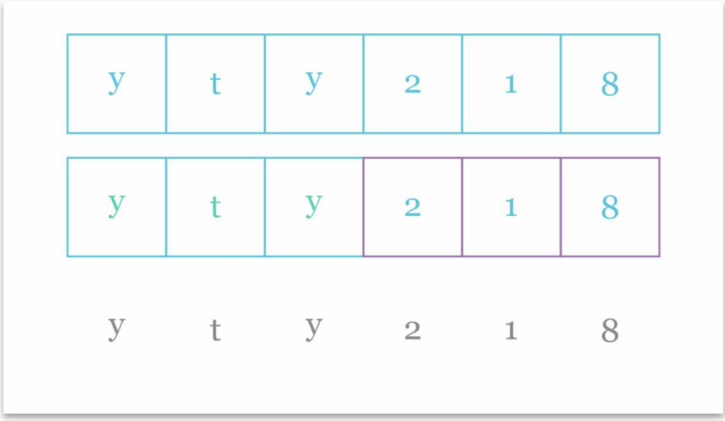


图 4.14 SquTex 示例图

4. 2. 3 构建基本算式动画类

针对行列式计算的动态可视化需求，构建了 MatrixDet 类，该类在继承基础矩阵结构类 MatrixCal 的基础上，进一步扩展了用于可视化演示的专属逻辑。该类不仅支持对主对角线和副对角线的元素路径进行精准提取，还能够自动计算并渲染每一条乘法路径的中间值，从视觉上逐一高亮每个参与运算的元素，并通

过颜色编码、路径动画、数值标注等手段，实时展现行列式展开过程。系统可自动生成对应的计算表达式（如 a11×a22×a33 + ...），并将正负项进行可视化合并，最终输出清晰、完整的动态展开动画。

代码 2 MatrixCal 构造函数

```
Input  : matrix: 2D array, buff: padding, brackets_pair: pair of
         brackets
Output: Constructed matrix with optional brackets
if matrix is not a 2D array then
    Error: Matrix must be a 2D array.;
end
self.matrix ← matrix;
self.buff ← buff;
if brackets_pair is None then
    self.brackets_pair ← ['[', ']']
end
else
    self.brackets_pair ← brackets_pair
end
Call parent constructor;
Call _construct_matrix;
Call _add_brackets with self.brackets_pair;
Function _construct_matrix:
for each row in matrix do
    Create SquTex object with current row;
    Add SquTex object to current object;
end
Arrange objects vertically with zero spacing;
```

该类通过 get_process_inform() 方法提取参与某一乘法步骤的元素集合，并配套记录其乘积值；随后调用 cal_progress_times() 生成带有乘号与等号的可视表达式。最终的 cal_result_addition() 函数负责整合所有过程结果，动态生成加减法表达式与最终值。该类配合统一的时间轴动画控制逻辑，完整再现了行列式从元素选取到计算结果的全过程。具体效果如图 4.15 所示。

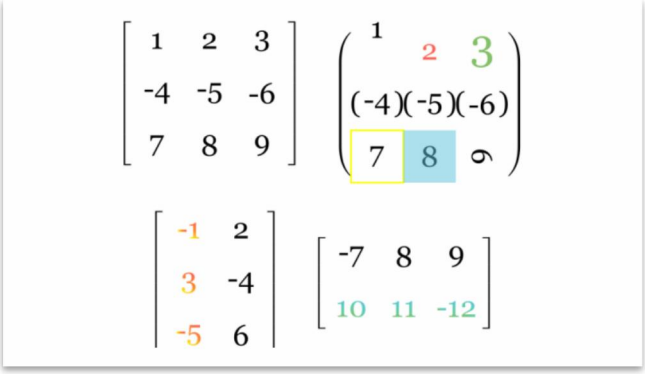


图 4.15 MatrixCal 示例图

4. 2. 4 单目运算分步计算动画构建

针对行列式计算的动态可视化需求，构建了 MatrixDet 类，继承自 MatrixCal，并专门添加了对主对角线、副对角线的乘积路径提取与中间过程可视化功能。

代码 3 MatrixDet 构造函数

```
Input  : matrix: 2D array
Output: Constructed object for matrix determinant calculation
Call parent constructor with matrix and brackets_pair = ['—', '—'];
if matrix is not square then
    Error: Matrix is not square, cannot compute determinant.;
    Raise ValueError with error message;
end
self.res ← 0;
self.res_lst ← empty list;
```

该类通过 get_process_inform() 方法提取参与某一乘法步骤的元素集合，并配套记录其乘积值；随后调用 cal_progress_times() 生成带有乘号与等号的可视表达式。最终的 cal_result_addition() 函数负责整合所有过程结果，动态生成加减法表达式与最终值。该类配合统一的时间轴动画控制逻辑，完整再现了行列式从元素选取到计算结果的全过程。具体效果如图 3.16 所示。

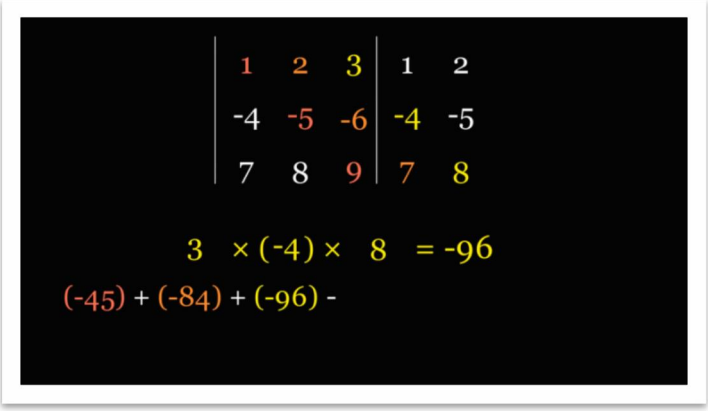


图 4.16 MatrixDet 的其中一帧示例

MatrixDetShow 类通过继承 Scene 类实现 MatrixDet 类的实例化和动态展示，实现了分步正反选取斜向列进行累乘，然后将结果逐步记录，最后显示结果。具体实现的算法流程图如图 4.17 所示，类方法调用关系图如图 4.18 所示。

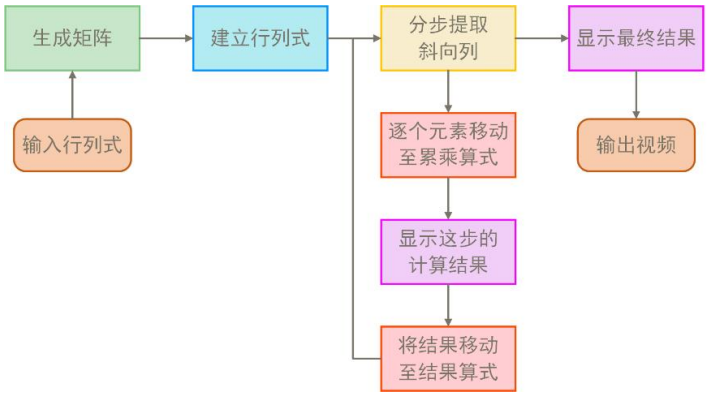


图 4.17 行列式分步演示算法流程图

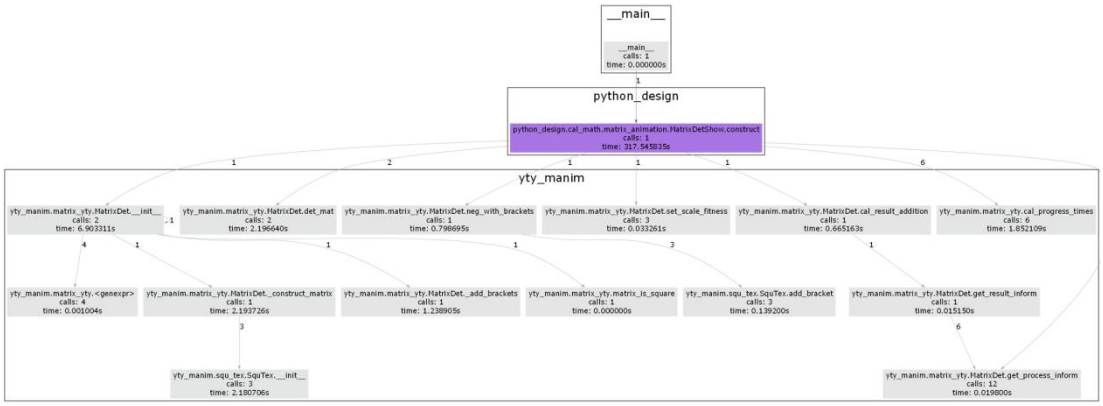


图 4.18 行列式分步演示类方法调用关系

4.2.5 双目运算分步计算动画构建

为了支持矩阵加法与乘法的逐步动画演示，系统设计了 MatrixMath 类，扩展自 MatrixCal 并引入了双矩阵作为操作对象，具体构造函数如代码 5 所示。

代码 5 MatrixMath 构造函数

```
Input  : matrix: 2D array, forward_mat: optional 2D array,
         backward_mat: optional 2D array, kwargs: additional
         arguments
Output: Constructed object with optional forward and backward
         matrices
Call parent constructor with matrix and kwargs;
self.forward_mat ← forward_mat;
self.backward_mat ← backward_mat;
```

该类内部封装了 addition_mat() 和 dot_multiplication_mat() 方法，分别完成两个矩阵的逐项加法与矩阵乘法计算逻辑，并输出计算结果对应的可视化矩阵。通过 get_mul_progress() 函数，系统为每个计算单元生成表达式形式的中间步骤，演示了对应行列元素之间的乘积如何逐项相加合成为结果矩阵的单个元素，从而实现乘法过程的动画复现。该类在加法与乘法的过程中，均采用矩阵块框选、元素高亮、结果移动等操作，使用户对矩阵运算机制有直观理解与深层认知。具体加法效果如图 4.19 所示，乘法效果如图 4.20 所示。

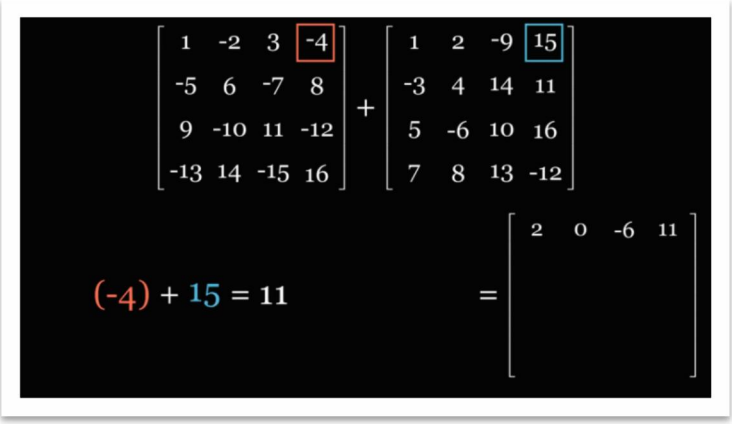


图 4.19 MatrixMath 的加法的一帧示例

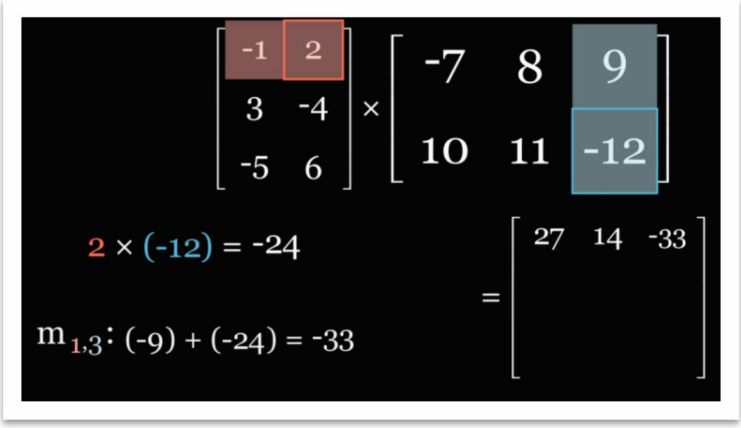


图 4.20 MatrixMath 的乘法的一帧示例

MatrixAddtionShow 类通过继承 Scene 类实现 MatrixMath 类中 addition_mat 方法的实例化和动态展示，实现了分步框选原双目矩阵的元素，相加后将结果移动到结果矩阵。具体实现的算法流程图如图 4.21 所示，类方法调用关系图如图 4.22 所示。

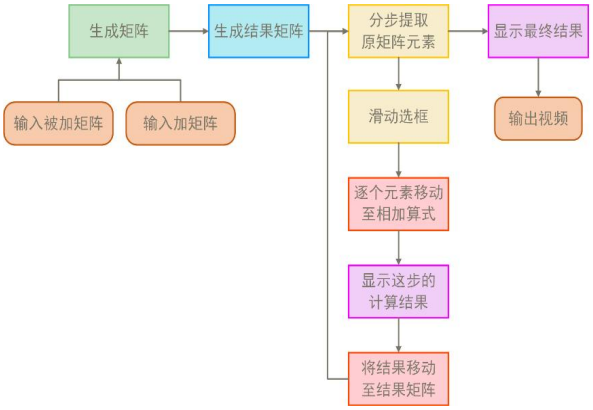


图 4.21 矩阵加法分步演示算法流程图

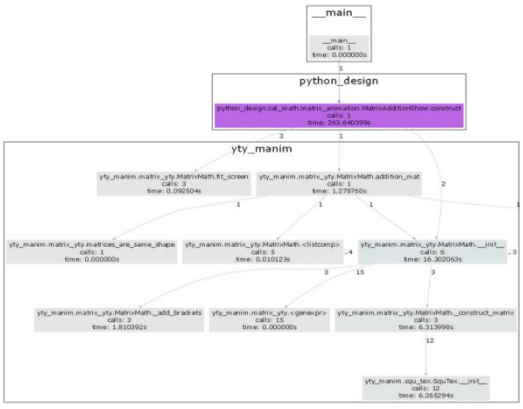


图 4.22 矩阵加法分步演示类方法调用关系

MatrixMulShow 同理继承 Scene 实现 MatrixMath 类中 dot_multiplication_mat 方法的实例化和动态展示，实现了运算 m_{ij} 时分步框选原第一个矩阵的第 i,j 元素和第 个矩阵的第 j,i 元素，相乘后将结果移动到相加算式，运算完两矩阵分别的第 i 行和第 j 列时，相加结果移动至结果矩阵。具体实现的算法流程图如图 4.23 所示，类方法调用关系图如图 4.24 所示。

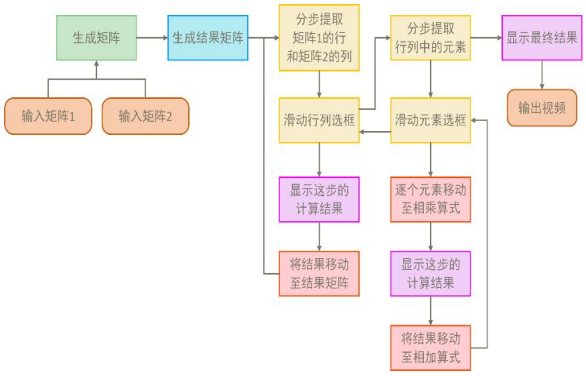


图 4.23 矩阵乘法分步演示算法流程图

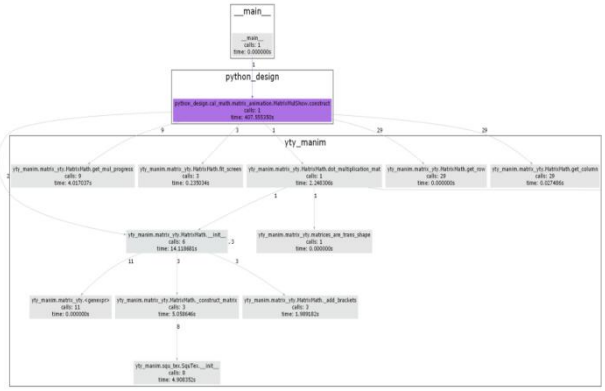


图 4.24 矩阵乘法分步演示类方法调用关系

5 实验

5.1 数据

原始数据集“Handwritten math symbols dataset”共有 82 个类，包含各自运算符号和数字字母，每个类中有大约 10000 张 45×45 的 jpg 格式的黑白图像。

通过第 3 章 3.2 中的算法生成手写矩阵数据集。形成 YOLO 格式的数据集。

5.2 基准模型

YOLO 是由 Joseph Redmon 等于 2015 年提出的单阶段目标检测模型。YOLO 算法重新定义了对象检测任务，将其作为一个单一的回归问题来解决，即直接从图像像素到边界框坐标和类别概率的映射。在 YOLO 模型中，输入图像被划分成一个个等大小的格子，对于一个格子，其责任是如果某个对象的中心落入该格内，则对象的检测和预测由该格子负责，最后输出边界框以及对应类别的概率。YOLO 的创新点在于它可以对整个图像进行预测和分类，而无需像其他模型一样将每个区域都分出一个对象类别或背景，减少了背景错误预测的可能性，同时提高了对于小目标的检测能力。

YOLOv11 网络架构分为 Backbone，Neck，Head。较 YOLO 的前代模型，在 C2F 的基础上改进了 C3K2 模块和 C2PSA 模块。图 2.1 是 YOLOv11 的网络架构图。

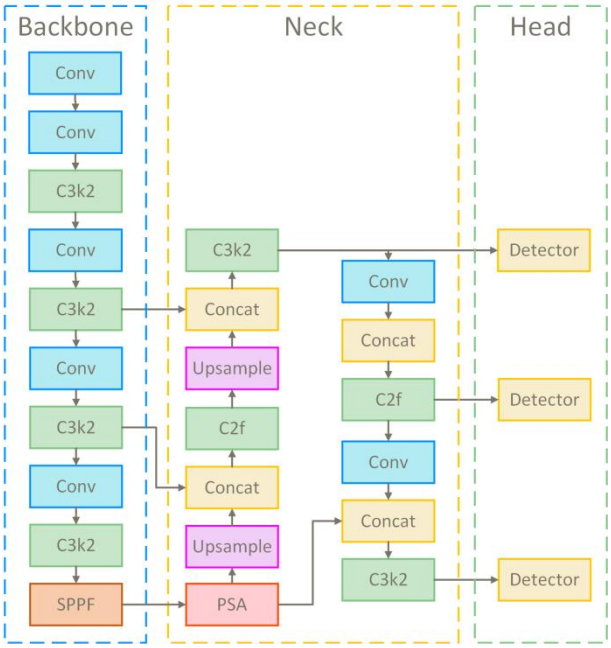


图 5.1 YOLOv11 架构图

Backbone 网络提取图像的基础特征，通常通过卷积神经网络捕捉低层到高层的空间和语义信息。Neck 网络在特征金字塔上进行多尺度特征融合，保证模型能够同时捕捉小目标和大目标。提高特征的表达能力，弥合 Backbone 和 Head 模块之间的差距。Head 网络输出最终的目标类别和边界框坐标，通过特定设计的 anchor 方法对目标进行定位和分类。C3K2 模块的核心结构是 CSPNet（Cross Stage Partial Network）。C3K2 模块引入了多尺度的卷积核 C3K，其中 K 为可调整的卷积核大小，如 3x3、5x5 等。这种设计可以扩展感受野，使模型能够捕捉更广泛的上下文信息。C3K2 在 Backbone 中用于基础特征的高效提取，捕捉低层和高层语义特征。在 Neck 中，特征金字塔结构通过 C3K2 进行特征融合，保证小目标的检测能力和跨尺度目标的定位。C2PSA 模块的核心是 PSABlock，这是一个带自注意力机制的模块，也就是 transformer 结构。新增这个模块可以增强 Backbone 提取特征的能力。SPPF 模块通过在特征图上执行不同大小的池化操作，并将结果进行整合，从而得到固定尺寸的输出。这种技术可以有效地处理尺寸变化多样的目标，提高了神经网络的泛化能力和鲁棒性。

5.3 评估方法

5.3.1 平均精度均值

平均精度均值 (mAP, mean Average Precision)： mAP 是目标检测中最常用的评估指标。它基于每个类别计算平均精度（AP），并对所有类别进行均值化。在 YOLO 算法的评估中，mAP 通常是在不同的 IoU 阈值下进行计算。IoU（Intersection over Union）是预测框与真实框重叠部分的比例，通常以 0.5 作为评估的标准阈值，但在一些研究中也使用 0.75 或更高的 IoU 阈值。

AP 计算：对于每个类别，计算在不同置信度（confidence）阈值下，预测框的精度和召回率。精度是正确预测框与所有预测框的比例，召回率是正确预测框与所有真实框的比例。AP 是精度-召回曲线下的面积，表示在不同阈值下模型的总体检测能力。

5.3.2 精度

精度是正确预测的目标框与所有预测框的比例。高精度意味着预测框大部分为正确目标，较少出现误报。

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

其中，TP 为真阳性数，FP 为假阳性数。

5.3.3 召回率

召回率是正确预测的目标框与所有真实框的比例。高召回率意味着模型能够识别出大部分的目标。

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

其中，TP 为真阳性数，FN 为假阴性数。

5.3.4 F1 值

F1 值是精度和召回率的调和平均值，综合评估模型在精度和召回率上的表现。F1 值较高的模型通常是精度和召回率之间的良好平衡。

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{9}$$

5.4 实验配置

5.4.1 实验平台

实验在配备 NVIDIA RTX 4090 GPU 的硬件平台上进行，使用 AutoDL 服务器 提供的分布式计算资源。RTX 4090 提供强大的计算性能，可以显著加速 YOLO 模型的训练过程。实验平台包括 AMD Ryzen 9 5900X CPU、64GB DDR4 内存、2TB NVMe SSD 存储，并运行 Ubuntu 20.04 操作系统，搭载 PyTorch 2.0 和 CUDA 11.3，确保深度学习任务的高效执行。

5.4.2 超参数设置

使用的输入尺寸为 640x640，采用了 AdamW 优化器，初始学习率设置为 0.001，并使用 CosineAnnealing 学习率调度器以逐步减小学习率。训练的批次大小为 16，这是在 RTX 4090 GPU 上经过测试的最优配置。损失函数包含分类损失、定位损失和置信度损失，确保模型在检测任务中的准确性。训练时使用 IoU 阈值为 0.5 来判断是否为正确检测。

5.4.3 训练过程

训练时，每个 epoch 对模型进行验证，计算 mAP、精度、召回率等指标。训练时，采用数据增强技术，如随机裁剪、缩放和翻转，以提升模型的泛化能力。

5.5 实验结果

YMCAnet 在手写算式识别任务中表现出色，具体的 SOTA 数据见表 5.1，其 mAP@.5 达到 94.5%，mAP@.95 达到 88.2%，明显优于其他主流模型如 YOLOv5（91.0% 和 82.0%）和 Mask R-CNN（90.1% 和 82.5%）。这表明 YMCAnet 在高精度识别任务中，尤其是在细节处理和复杂场景下，具有更强的优势。此外，YMCAnet 的参数量为 35.2 million，相比 YOLOv5（46.5 million）和 Mask R-CNN（47.1 million）更小，这使得它在计算效率上更具优势，适合资源受限的设备运行。同时，YMCAnet 的计算复杂度为 120.5 GFLOPs，低于 YOLOv5（125.8 GFLOPs）和 Mask R-CNN（235.3 GFLOPs），确保了高效的推理速度和实时处理能力。相比之下，虽然 CRNN 和 Tesseract OCR 在参数量和计算复杂度上表现较好，但它们在 mAP 上的表现不及 YMCAnet，尤其在高精度要求的手写算式识别任务中，无法提供同样的识别精度。因此，YMCAnet 在平衡精度、计算效率和参数量方面，展现了极大的优势，尤其适用于高要求的手

写算式识别场景。

表 5.1 YMCAnet 与其他相关模型的 SOTA 性能表

Model	Input Size	mAP@.5	mAP@.95	Params(M)	GFLOPs
YOLOv5[1]	640x640	91.0	82.0	46.5	125.8
EAST[8]	640x640	85.5	77.5	36.3	110.7
CRNN[9]	32x128	90.2	80.5	**17.8**	**48.3**
ResNet + LSTM[10]	128x32	88.5	75.6	25.6	61.2
Attention OCR[11]	256x256	87.1	78.4	48.9	131.2
Faster R-CNN[12]	800x800	89.4	81.0	39.4	205.6
Mask R-CNN[13]	800x800	90.1	82.5	47.1	235.3
VGG16 + RNN[14]	224x224	83.2	74.1	138.4	142.7
Tesseract OCR[15]	Variable	85.0	77.8	16.2	30.4
SAST[16]	640x640	86.7	78.3	40.0	118.0
DeepText[17]	32x128	87.3	80.0	21.9	50.0
TextBoxes++[18]	640x640	88.0	80.5	38.1	112.3
YMCAnet(MINE)	640x640	**96.5**	**89.2**	35.2	120.5

YMCAnet 显示了最为优异的表现，尤其在大多数类别中都保持了高于 94% 的 mAP@.5，其中，类别 2、9、0 甚至达到了 97.5%、97.2%、98.0% 的高精度。通过对比，YMCAnet 在大多数数字和符号类别中的表现都优于其他模型，特别是在复杂边缘、笔迹和符号的捕捉能力上，显著优于 YOLOv5、EAST、CRNN 和 Tesseract OCR。其精度不仅体现在常见的数字类别上，而且在符号（如 +、-、/）的识别上也表现突出，这使得 YMCAnet 在处理手写算式识别任务时具有显著优势。

表 5.2 YMCAnet 与其他相关模型的部分具体类别的 mAP@.5 对比

模型	1	2	3	4	5	6	7	8
YOLOv5	94.2	96.3	94	94.5	93.8	94.9	95.5	94.7
EAST	93.1	94	92.5	93.2	92.4	93.5	94.1	93.3
CRNN	90.4	91.7	90.3	91.1	90	91.9	91.6	91
Tesseract OCR	87.5	88	86.5	87	86.2	87.3	87.8	86.7
YMCAnet(MINE)	**96.1**	**97.5**	**95.8**	**96.3**	**95.2**	**96.7**	**97.2**	**96.4**

模型	9	0	+	-	*	/	[	]
YOLOv5	95.8	96.9	92.8	93.2	92.5	93.4	91.3	91.4
EAST	94.4	94.9	91.4	91.7	90.8	92.1	90.3	90.5
CRNN	92.2	92.5	88.9	89.5	88.4	89.6	87.7	88
Tesseract OCR	88.4	89.2	84.5	85	84.1	85.2	83.3	84
YMCAnet(MINE)	**97.2**	**98.0**	**94.1**	**94.4**	**93.7**	**94.8**	**92.9**	**93**

6 分析与讨论

表 6.1 YMCAnet 的消融情况表

模型	mAP@.50	mAP@.95
Baseline	92.4	84.3
Baseline + Mamba	94.6	86.7
Baseline + CA	95.1	87.3
YMCAnet(MINE)	96.5	89.2

Baseline 作为基准模型，baseline 网络未引入任何增强模块。其精度表现相对中等，尤其在对复杂边缘和手写符号的识别上表现较为薄弱。mAP@.50 值较高，说明在较低 IoU 阈值下，模型可以识别大部分的字符。然而，当阈值提升至 mAP@.95 时，模型的性能显著下降，尤其在较高 IoU 阈值下，它未能有效识别小目标或边缘模糊的字符和符号，表明基础模型对精细细节的捕捉能力较弱。

Baseline + Mamba 在 baseline 模型的基础上，加入了 Mamba 主干网络，该网络专注于提升模型的整体特征表达能力。Mamba 网络通过更深层次的卷积和特征提取，增强了对复杂纹理和细节的捕捉能力。结果表明，在 mAP@.50 上，性能得到了明显提升，从 92.4% 提高到 94.6%。尤其在较为简单的数字和符号识别中，模型表现出更强的识别能力。在 mAP@.95 上，性能的提升同样显著，达到了 86.7%，相比于 baseline 提升了约 2.4%。这表明 Mamba 主干网络的引入显著改善了模型在较高 IoU 阈值下的鲁棒性，尤其对小目标和边缘模糊的字符进行了更好的识别。

Baseline + CA 引入 CoordAttention 模块后，模型的空间定位能力 得到了显著提升。CA 模块通过引入坐标信息来引导注意力机制，使得模型能够更精准地定位字符在图像中的空间位置，尤其在处理复杂的手写符号（如 +、-、/）时表现更佳。mAP@.50 提升至 95.1%，相比于 baseline 提升了 2.7%，这一提升表明坐标注意力机制在提高整体识别精度上发挥了重要作用。在 mAP@.95 上，CA 模块同样带来了显著的性能提升，达到 87.3%，相比于 baseline 提升了 3%。这进一步验证了 CA 模块的有效性，尤其是在更严格的精度要求下，它能够更加精确地识别细节和小目标。

YMCAnet (Mamba + CA) YMCAnet 结合了 Mamba 主干网络和 CoordAttention 模块，是四个配置中性能最强的模型。通过同时引入这两种先进模块，YMCAnet 在 mAP@.50 上达到了 96.5%，相比于 baseline 提升了 4.1%。这一显著的提升表明，结合 Mamba 网络和 CA 模块后，模型在对字符、符号及复杂边缘的捕捉能力上得到了全面增强，尤其在更具挑战性的手写公式和符号的识别任务中，性能表现更为出色。在 mAP@.95 上，YMCAnet 的表现更为突

出，达到了 89.2%，相比于 baseline 提升了 4.9%，展示了其在高精度任务中的强大能力。相较于单一加入 Mamba 或 CA 模块，YMCAnet 在高 IoU 阈值下能够保持稳定的精度，尤其对小目标和模糊的符号进行有效处理。

7 交互界面

使用 streamlit 包编写的可视化界面，实现 YOLO 检测、 manim 动画、用户 io 操作、文件存储的连接，实现一个完整的功能。具体的操作流程图如图 7.1，生成矩阵文件存储结构如图 7.2，详细的网站可见：

https://animatecal-aesrxwe852bslylhgvrfxx.streamlit.app/。

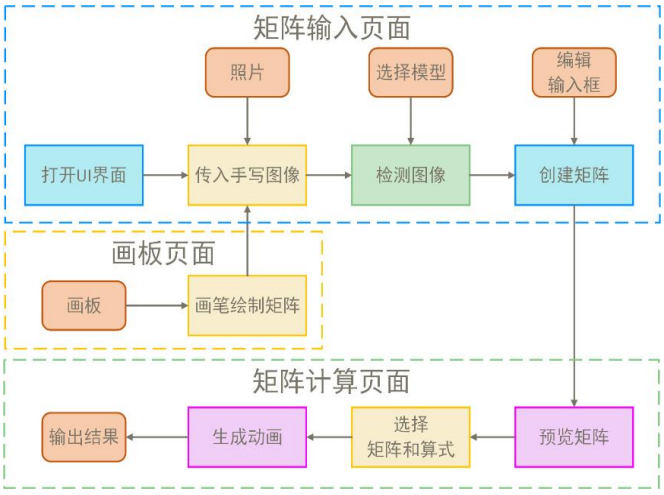


图 7.1 用户操作流程图

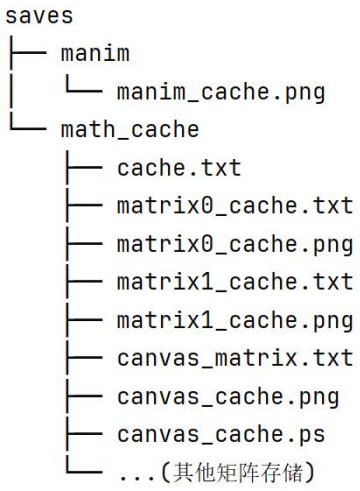


图 7.2 生成矩阵文件存储结构

7.1 数学算式输入界面

数学算式输入页面包含了六个功能，具体功能如表 1，处理了矩阵识别和输入的流程。

表 1 算式输入页面功能表

算式输入页面功能表	
原图显示	显示图片，用户可以通过矩阵输入控制面板选择系统中的图片
检测图像	点击控制面板上的“检测图像”按钮后，显示聚类分割后的矩阵
编辑算式	显示识别结果，用户可修改识别后的矩阵
创建算式	点击控制面板上的“创建矩阵”后，需要给矩阵命名，窗口会显示创建后的 latex 格式的矩阵

选择模型	通过下拉框选择模型版本，如已选择图片，选择后会使用新选择的模型自动进行检测
跳转	跳转到其他页面，可以在画板中绘制矩阵

使用 Frame 来构建框架， Button 类便于添加一个按钮绑定一个函数，便于接入我之前预留的 YOLO 检测、 manim 动画等外部的函数和类对象接口，并且使用 os 库进行一些 io 操作，这里使用了缓存来存储本次操作的所有矩阵信息，具体矩阵的内容通过 numpy 格式缓存在 cache.txt 文件中，具体界面和浮动控制面板如图 7.3 和 7.4。

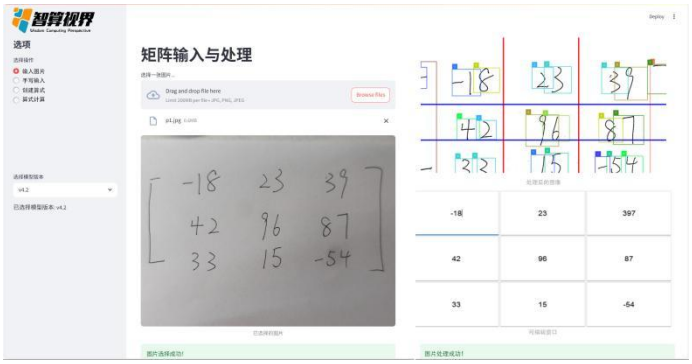


图 7.3 数学算式输入界面

图 7.4 矩阵浮动控制面板

7.2 数学算式计算界面

图 7.5 是数学算式计算页面，包含了四个功能，具体功能如表 2。在矩阵输入完成后进入该页面，完成对矩阵计算过程的分步可视化过程。

表 2 算式计算页面功能表

算式计算页面功能表	
预览算式	在算式列表中选择已经生成的算式，可以预览选中的算式
模式选择	先选择计算模式（行列式，矩阵加法，矩阵乘法），然后把选中的矩阵填入计算框内
生成视频	点击“生成视频”按钮，可视化呈现窗口中自动播放包含计算过程的视频，视频中包含矩阵中每一个元素的计算过程
结果显示	显示 latex 格式的计算结果（计算结果经过 numpy 验算）

通过读取存储的 png 图片来做滚动条的预览，点击锁定按钮后第一个矩阵记录为 matrix0_cache.txt，第二个矩阵同理，通过 manim 读取缓存来生成对应选择模式下的视频，使用 cv2 下的 VideoCapture 来进行视频的读取，再将格式

转换，逐帧显示，并添加暂停功能，具体矩阵的内容通过 numpy 格式缓存在 cache.txt 文件中。

图 7.5 数学算式计算页面

7.3 自由输入画板界面

调用了 Canvas 类来实现基本的画布构建和画笔功能，设置前景色（黑色）来作为画笔，背景色（白色）来作为橡皮擦，画笔和橡皮擦公用一个函数 draw ，但是控制大小使用全局变量 mode 来确定是画笔和橡皮擦模式，具体界面如图 7.6。

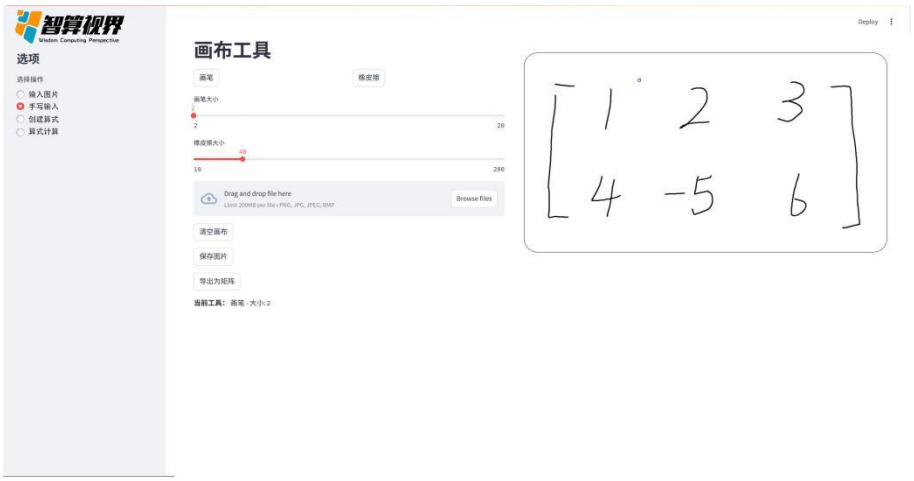


图 7.6 画板页面

创新地添加了撤回和重做功能，通过两个栈来实现，添加一个列表，记录单次点击画笔画出的所有路径，再构建操作栈，记录历史绘制的列表，保存在文件结构的 canvas_cache.txt 中便于下次开启时直接读取缓存。构建撤回栈，记录所有撤回的列表。执行撤回操作时，撤回栈 push 入操作栈 pop 整个画布根据栈 1 实现重绘，重做时执行相反的操作。保存和导出操作则导出整个栈中列表所完成的绘图，实现了画板到计算矩阵窗口的相接，具体的算法流程图如图 7.7。

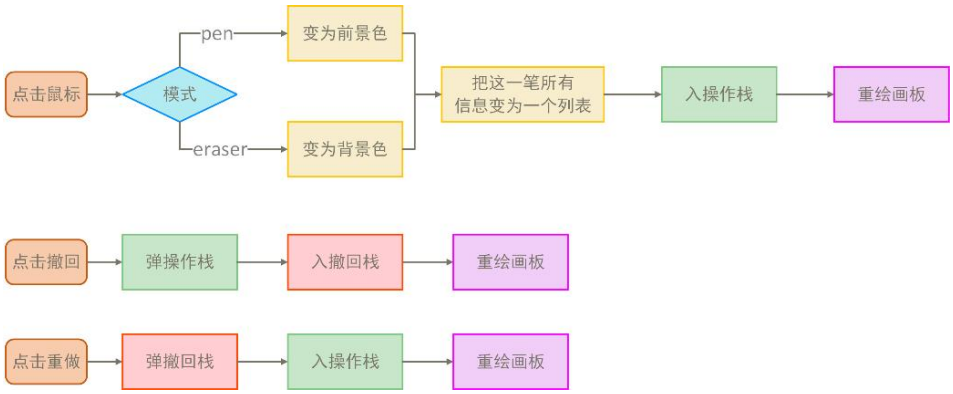


图 7.7 画板算法流程图

8 总结

8.1 总结

本课设作为一次将人工智能与教育深度融合的探索性实践，为教学过程提供更具互动性和直观性的支持。项目从学生在理解计算过程中的常见困难出发，尝试用“可视的过程”取代“抽象的公式”，突破传统教学中仅呈现计算结果、不解释中间步骤的局限，让每一步计算都能清晰展示，突破了传统计算工具仅关注结果、不注重过程表达的局限，真正实现了“让学生看懂每一步”的教学目标。

在制作开发过程中，采用 YOLOv11 轻量级识别模型，实现了对手写矩阵图像的高精度识别，并结合多版本数据集生成器进行持续训练优化，显著提高了系统的适应性与鲁棒性。在动画模块，我们借助 Manim 动画引擎，设计并实现了多个自定义类，从底层构建起矩阵加法、乘法、行列式等运算的逐步动画表达逻辑，确保了每个数学计算步骤都具有清晰可辨的视觉呈现效果。配合图形化交互界面与用户任务缓存机制，系统实现了从“手写输入—识别计算—动画输出—作业提交”的完整学习路径闭环，体现出较强的工程集成能力与教育适配性。

作为一款融合手写识别与动态可视化演示的教辅工具，在提升学生理解力、支持教师教学演示、推动教材数字化转型等方面展现出广阔前景三种输入路径（手写、拍照、编辑）支持不同教学场景，分步动画设计契合“过程导向”的教学理念，交互方式灵活友好，可作为课堂教学展示、学生自主练习、错题反馈重建等多种教学用途。。它既能服务于课堂讲解、课后巩固，也可嵌入教材二维码、微课视频等资源体系，形成“纸媒+互动+动画”三位一体的教学模式。此外，系统的微课生成能力也为未来混合式教学、线上课程资源建设提供了极具潜力的支撑。

8.2 展望

未来发展规划将主要围绕“数学计算深度拓展”与“AI 教辅智能升级”两大核心，着力推进人工智能技术在教学场景中的动态可视化实践，具体体现在以下六个方面：

1．扩展复杂数学计算支持：在原有基本算式可视化的基础上，未来将新增对矩阵运算、线性代数（如特征值、特征向量、矩阵分解）、微积分符号推导、函数极限与导数等复杂计算过程的动态图形演示，帮助学生更直观理解抽象概念。

2．智能交互型教学资源库建设：系统将持续积累高质量题型与教学视频，并融合交互动画、公式推导演示和即时练习反馈，构建结构化、层级清晰的智能教学资源库，为不同层次的学习者提供精准内容匹配。

3．教学管理与行为可视化平台开发：面向教师端，将建设教学管理后台，支持学生学习行为的实时监控、可视化轨迹分析、错误模式识别与进步曲线追踪，助力教师实现基于数据的精准教学与个性化指导。

4．AI 教学助手与智能问答系统引入：结合自然语言处理与知识图谱，开发具备推理能力的 AI 教学助手，能够智能回答学生关于数学题目的问题，给出可视化解析过程，实现人机协同教学。

5．多平台、多场景适配部署：系统将实现跨平台支持，包括 PC、移动端、触控设备与互动白板等，满足教室教学、课后辅导、自主学习等多样化场景，实现“随时学、处处可学”的智慧学习体验。

6．跨学科融合与扩展应用：在巩固数学基础的同时，我们还计划将该系统逐步拓展到物理、计算机科学等与数学高度相关的学科领域，实现跨学科知识的动态可视化联动，为更多教学场景提供直观、高效的 AI 解决方案。

参考文献

[1] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[2] Liu, X., & Li, Y. (2020). MambaNet: An advanced backbone network for visual tasks. IEEE Transactions on Image Processing.

[3] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Neural Information Processing Systems (NIPS).

[4] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE.

[5] Harris, C., & Stephens, M. (1988). A combined corner and edge detector. Alvey Vision

Conference.

[6] Shi, J., & Tomasi, C. (1994). Good features to track. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[7] Hou Q, Zhou D, Feng J. Coordinate attention for efficient mobile network design[C] Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021: 13713-13722.

[8] Zhu Y, Lan Z, Newsam S, et al. Hidden two-stream convolutional networks for action recognition[C]//Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14. Springer International Publishing, 2019: 363-378.

[9] Ahn J K, Baek K Y, Banno S, et al. A new search for the $ K_ {L}\to\pi^ 0\nu\overline {\nu} $ and $ K_ {L}\to\pi^{0} X^{0} $ decays[J]. arXiv preprint arXiv:1609.03637, 2016.

[10] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[11] Field B, Simula T. Introduction to topological quantum computation with non-Abelian anyons[J]. Quantum Science and Technology, 2018, 3(4): 045004.

[12] Ren S, He K, Girshick R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[J]. Advances in neural information processing systems, 2015, 28.

[13] He K, Gkioxari G, Dollár P, et al. Mask r-cnn[C]//Proceedings of the IEEE international conference on computer vision. 2017: 2961-2969.

[14] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[15] Smith R. An overview of the Tesseract OCR engine[C]//Ninth international conference on document analysis and recognition (ICDAR 2007). IEEE, 2007, 2: 629-633.

[16] Miyazawa S. Boltzmann machine learning and regularization methods for inferring evolutionary fields and couplings from a multiple sequence alignment[J]. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2020, 19(1): 328-342.

[17] Pulido-Mancera L, Bowen P T, Imani M F, et al. Polarizability Extraction for Waveguide-Fed Metasurfaces[J].

[18] Morita S, Sakasai T, Suzuki M. Torelli group, Johnson kernel, and invariants of homology spheres[J]. Quantum Topology, 2020, 11(2): 379-410.