

# 华南理工大学

## 《神经网络与深度学习》课程实验报告

实验题目： Image-to-latex-main, 手写图片转 latex 公式

姓名： 张杰铖 学号： 202030443394

班级： 计科全英联合班 组别： 40

指导教师： 余志文

### 实验概述

#### 【实验目的及要求】

实验目的：

通过实践检验学习成果。在之前的作业当中，我们已经熟悉了深度学习的基本原理和 Pytorch/TensorFlow 框架的使用方法，但是之前的作业距离实际的开发或科研仍然相差甚远。为了检验我们自己的学习成果，我们以小组的形式合作从数据标注开始，最终训练一个手写公式识别模型。

实验要求：

1. 熟悉并使用 labelImg 软件提取公式图片。本次实验会提供真实的初高中数学试卷作为数据源给每位同学，每位同学负责其中一部分图片的公式框选。（步骤一）
2. 待每位同学完成后，将会收集同学们框选的标注，通过 mathpix 识别后，取 mathpix 的识别结果作为 ground truth，再发回给大家作为数据集来训练。（步骤二）
3. 下载获取数据集（步骤三）
4. 利用所提供的代码，完成数据的清洗+预处理工作，以便作为模型的输入。（步骤四）
5. 训练两个模型：（步骤五）  
模型 1：Encoder 用 CNN，Decoder 用 RNN  
模型 2：Encoder 用 Resnet，Decoder 用 Transformer
6. 提交模型测试集预测结果，获取测试集成绩（步骤六）
7. 准备小组答辩，同时提交实验报告和源代码。（步骤七）

评分标准：

1. 数据标注（40 分）：高质量完成对应标注任务即可得到该部分分数的 85%，额外标注一份即可得到该部分分数的 100%。注：若标注质量低则酌情扣分。
2. 模型实现（50 分，结合答辩环节评估）：
3. 模型正确性（30 分）：模型 1 CNN+RNN（15 分）和 模型 2 Resnet+Transformer（15 分）。评分根据参考代码实现是否正确、实验结果评测指标、代码可复现性和注释等方面来考虑。
4. 模型拓展性（20 分）：优化模型、优化数据预处理、讨论任务瓶颈等。有 UI 的根据 UI 的美观、实用性等方面酌情加分。
5. 实验报告（10 分）：材料完整性，命名规范性，书写美观性等等。

## 【实验环境】

操作系统: Linux Ubuntu

## 实验内容

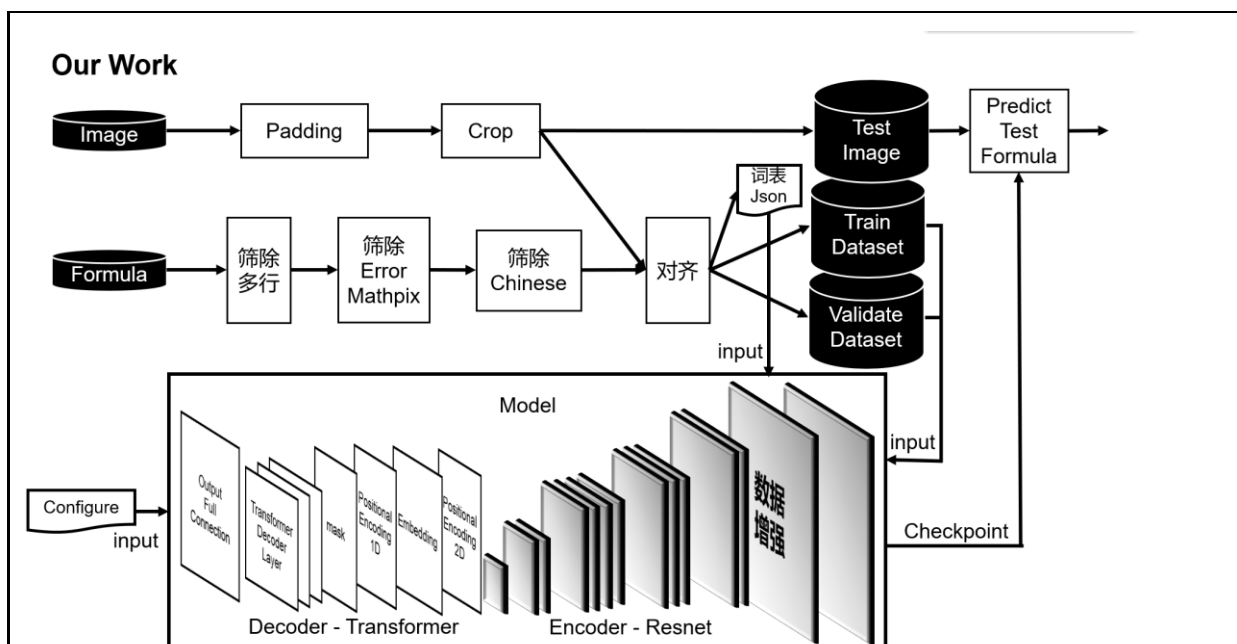
### 【实验过程】

1. 数据标注:  
标注采用 labelImg, 其是目标检查与识别科研界内广泛使用的开源标注软件。按照教程安装熟悉软件, 并下载所需要标注的试卷后便可以开始严格按照要求标注公式。标注过程略。
2. 等待助教整理收集标注图片, 用 mathpix 转化为公式后上传数据集。
3. 下载数据集。
4. 分工, 着手数据预处理与模型搭建。

组员姓名	组员工作
任雪卉	数据预处理, 增加数据集, 跑模型 1 数据, 解决测试集乱序问题, 提出优化模型 1 想法, 跑模型 2 数据, 测试模型, 制作 PPT。
欧阳欣平	搭建环境, 数据预处理, 尝试 Pytorch 模型, 跑模型 1 数据, 解决测试集乱序问题, 提出优化模型 1 想法, 测试模型, 制作 PPT, 答辩。
何其嫣	搭建环境, 数据预处理, 增加 Latex 字典, 尝试 Pytorch 模型, 跑模型 1 数据, 优化模型 1、模型 2 的超参数, 跑模型 2 数据, 测试模型, 制作 PPT。
林秀丹	搭建环境, 尝试 Pytorch 模型, 数据预处理, 跑模型 1 数据, 绘制对比图像, 实验结果的对比分析并给出改进意见, 测试模型, 制作 PPT。
张杰铖	搭建实验环境, 数据预处理, 优化模型 2 超参数, 模型 2 结构的优化探索, 数据增强, 结果预测和性能评估分析, PPT 制作, 答辩。

由以上分工可得, 我在小组内主要参与完成的是模型二搭建优化的全过程。

以下为我在答辩 PPT 中制作的模型二的流程图, 可以清晰可见我们工作的全流程:



对原始的数据集图像：

1. 无论是 train, dev 亦或是 test, 我都进行了 padding 操作。因为在试卷中标注的公式很难保证边缘有一定留白, 大多数公式都存在靠近边界的情况, 采用 padding 操作有助于保留边界信息。因为如果不加上 padding, 边界限速点可能只被计算一次, 而加上后, 会被多次计算。

Padding 操作的代码使用的是助教提供的代码, 其会将图片 padding 至最靠近 (略大于) 其宽、高的像素的 buckets。

```

buckets = [
    [60, 40], [80, 40], [80, 60], [140, 60], [140, 80],
    [240, 100], [320, 80], [400, 80], [400, 100], [480, 80], [480, 100],
    [560, 80], [560, 100], [640, 80], [640, 100], [720, 80], [720, 100],
    [720, 120], [720, 200], [800, 100], [800, 320], [1000, 200],
    [1000, 400], [1200, 200], [1600, 200], [1600, 1600]
]
  
```

2. 在 padding 操作后, 我又进行了裁剪(crop)操作。因为 padding 实际不是完全对称地进行 padding, 可能造成公式不在画面中心的情况。Crop 有助于让公式在画面上的位置更显著, 更易于被机器识别; 能减少图片的大小, 节省显存空间; 同时还能主观地保留一定量的空白边。

Crop 的代码为 Image-to-latex-main 项目中 scripts 文件夹内的 prepare\_data.py 以及与之相关的 image-to-latex 文件夹中 data 文件的 utils.py。

裁剪发现, 有几张标注图片不含公式:

Cropping images...

81305.png is ignored because it does not contain any text

81329.png is ignored because it does not contain any text



88194.png is ignored because it does not contain any text

查询发现, 其均存在于 test 测试集中。

对原始的数据集公式：

1. 采用助教所提供的 data\_filter.py 对 train、dev 的公式文件进行清洗, 先后筛除具有多行公式的文





件以及存在有 error mathpix 的文件。

 510.txt - 记事本 文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H) $\begin{array}{l} \lambda=2 \\ \lambda=0 \\ \lambda=0 \end{array}$	 1226.txt - 记事本 文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H) error mathpix
多行的公式	Error mathpix

2. 采用助教所提供的 no\_chinese.py 分别对 train、dev 和 test 中的公式进行词表的匹配。如果公式中的字符不存在于此表 vocab.json 中，则视为中文符号，该公式被筛除。例如：

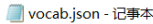
$\triangle D \perp \text{text} \{ \text{留} \} P C D$        $\text{text} \{ B c \cup p \}$

分别对原始数据集的图像和公式进行清洗和处理后，利用助教所提供的 shuffle\_and\_build\_dataset.py 进行对齐和数据集的搭建。对齐的目的是筛除那些已经但仅在图像集或公式集中被筛除的另一对应文件。该算法的初始代码是将所有图像和公式按照命名对齐，形成一整个数据集，将其按比例划分为训练集、验证集、测试集三部分，综合信息得到 im2latex\_formulas.norm.lst, im2latex\_train\_filter.lst, im2latex\_validate\_filter.lst, im2latex\_test\_filter.lst 这几个文件。而由于助教已经分好了特定的 train、dev、test，因此我们需要注释并重写部分代码，主要是保留对齐部分了生成综合信息文件。

 im2latex_test_filter.lst	2022-11-18 0:02	LST 文件	216 KB
 im2latex_formulas.norm.lst	2022-11-18 0:00	LST 文件	2,327 KB
 im2latex_validate_filter.lst	2022-11-18 0:00	LST 文件	213 KB
 im2latex_train_filter.lst	2022-11-17 23:58	LST 文件	1,052 KB

在此基础上，可以认为我们得到了划分并处理好的训练集、验证集、测试集了。其中测试集是缺少 ground truth，即公式的。

在 image-to-latex-main 项目的 scripts 文件夹中，prepare\_data.py 还通过系统调用执行 ssh 对 ground truth 文件 im2latex\_formulas.norm.lst 清洗得到 im2latex\_formulas\_norm.new.lst 文件，值得注意的是这需要在 Linux ubuntu 上将 shell 切换为 dash 才能成功执行。以及通过调用 image-to-latex 文件夹 data 文件夹中 utils.py 的分词器来新建词表 vocab.json。

 vocab.json - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
{"<PAD>": 0, "<SOS>": 1, "<EOS>": 2, "<UNK>": 3, "m": 4, "^": 5, "{": 6, "3": 7, "}")": 8, "-": 9, "4": 10, "e": 11, "4, "l": 95, "\hbar": 96, "i": 97, "\left": 98, "\right": 99, "\prime": 100, "v": 101, "r": 102, "o": 103, "S": 104, "stackrel": 167, "\zeta": 168, "?": 169, "K": 170, "\rho": 171, "\widehat": 172, "\nabla": 173, "&": 174, "\int":

初始的时候我们想对词表进行一个补充和纠正，因为里面存在某些错误，较为明显的诸如  $\angle$  变为了  $\text{angle}$ 。但是思虑再三，我们认为对其进行简单的补充和纠正反而可能让其准确率下降。

毕竟词表已经包含了 mathpix 所生成的所有公式里的词，我们的模型实质上更像是去靠近 mathpix 这个软件里模型预测的结果，可见本项目存在一定的局限性，模型效果的好坏最终在实际应用中并不一定准确。

以上为数据的预处理部分，以下开始模型的搭建

对于实验的硬件配置，我基本选择的都是 AutoDL 或者恒源云这两个算力平台的云主机，显存高达 24G 的 3090 显卡为主力军，操作系统为 Linux 的 ubuntu。

环境的具体搭建和运行方式如下图一致，



```
指南.txt - 记事本
文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)
pip install或者conda install安装
将requirement.txt的库都安装了
将cuda和torch升级到对应显卡版本
库torchmetrics安装0.5.0版本
sudo dpkg-reconfigure dash 然后选 no 将命令行切换模式
库setuptools直接安装
库torchvisions安装最新版本
在终端输入wandb init, 注册并输入账号码, 选择默认的image-to-latex
修改run_experiment.py里面的sys路径到项目根目录
然后应该可以run_experiment.py了
运行完之后python scripts/download_checkpoint.py + wandb运行项目名称如 alex_zhang/image-to-latex/3anrk
pre.py文件, 要修改一下test_id的绝对路径和生成的result的绝对路径
最后得到result.txt文件
```

这是我作为组员们撰写的运行指南，方便他们上手模型二一起跑模型。

## 1. 模型的最佳宽度和深度的探索

从网上的资料来看，一般而言，神经网络模型越宽越深，性能就会越好。

模型的主体部分主要来自于 resnet\_transformer.py 中的这段代码：

```
# Encoder
resnet = torchvision.models.resnet34(pretrained=False)
resnet2 = torchvision.models.resnet18(pretrained=False)
self.backbone = nn.Sequential(
    resnet.conv1,
    resnet.bn1,
    resnet.relu,
    resnet.maxpool,
    resnet.layer1,
    resnet.layer2,
    resnet.layer3,
)
self.bottleneck = nn.Conv2d(256, self.d_model, 1)
self.image_positional_encoder = PositionalEncoding2D(self.d_model)

# Decoder
self.embedding = nn.Embedding(num_classes, self.d_model)
self.y_mask = generate_square_subsequent_mask(self.max_output_len)
self.word_positional_encoder = PositionalEncoding1D(self.d_model, max_len=self.max_output_len)
transformer_decoder_layer = nn.TransformerDecoderLayer(self.d_model, nhead, dim_feedforward, dropout)
self.transformer_decoder = nn.TransformerDecoder(transformer_decoder_layer, num_decoder_layers)
self.fc = nn.Linear(self.d_model, num_classes)
```

其中，涉及到宽度和深度的部分主要有：

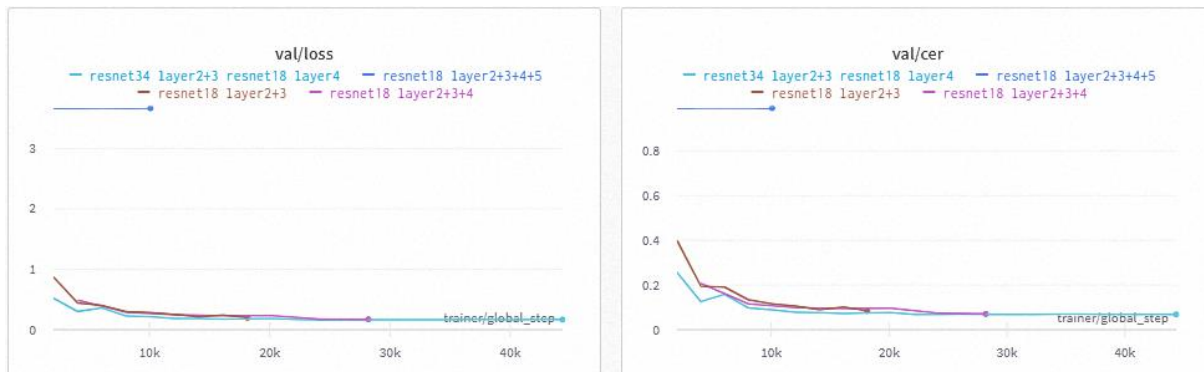
Resnet 中的 backbone 模块，其原本为 Resnet 18 的 conv1 至 conv4\_x，即先卷积池化再陆续经过 3 个 block。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

Transformer 中的 Dimension of feed forward, Number of decoder layers, Number of head。

整个模型上的 Dimension of model, Output max length。

一般而言，Output max length 不修改。我首先采用控制变量法地尝试了 Dimension of feed forward, Number of decoder layers, Dimension of model 这几个参数的增大，以及将 resnet 中的 backbone 模块调整至包含 conv1 至 conv5\_x，但实际测试加宽加深模型的效果均不佳，无论是 train loss 或者是 val loss 均维持在 3.6 左右无法下降。我尝试调小学习率以及减小 batch size 也没有起到任何效果。以下的图是原模型和加宽加深模型的对比，我仅仅保留了将 resnet 中的 backbone 模块调整至包含 conv1 至 conv5\_x 的测试曲线作为代表。



别的参数不谈，理论上至少 resnet 的加深不应引起网络的退化，我百思不得其解。查阅资料，我猜测出现这种情况的原因是加宽加深的模型获取到模型的特征太冗余了、太浓缩了。如 resnet 的 backbone 在增加 resnet 18 的 conv5\_x 时再一次对图片进行了卷积等操作，将 256 的通道扩展至了多达 512 的通道，size 也缩小一倍。我们输入的手写公式图片的特征可能并不需要那么多，卷积过多反而让一些特征变得模糊、混合，难以精准预测单个字符。并且从性能表现看，加宽加深的模型耗费的计算资源指数级增长，这是不值得的。

在此猜想下，我认为模型至少不应再加宽。

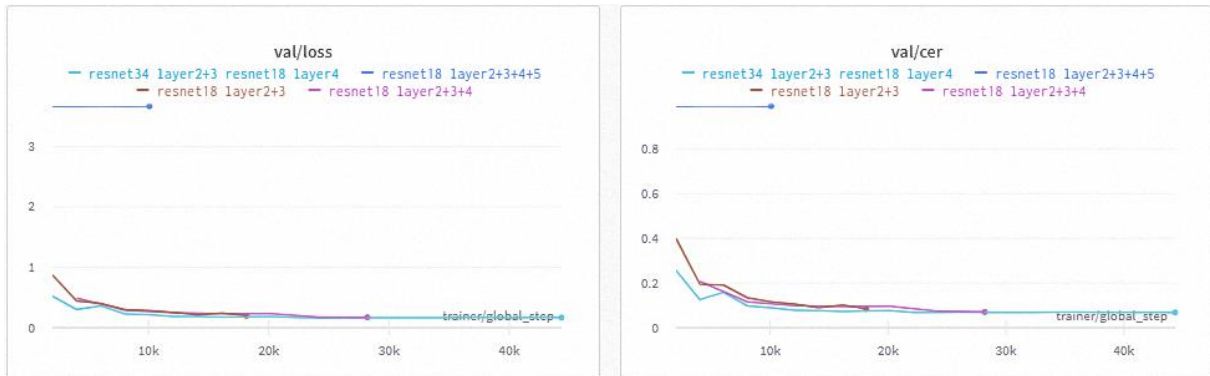
我又尝试了一下减少模型的宽度和深度，即将 resnet 的 backbone 的 conv4\_x 删去。如上图曲线所描述，令人惊奇的是其性能和未删除的情况很接近，仅在 val/cer 指标上略逊一筹。但考虑到我们拥有的算力资源无需如此简朴地采用这么浅的模型，因此我们又回到了原始模型。

经过以上思考后，我尝试略微加深模型。此时我的选择是修改 resnet 的 backbone 中的结构，将 resnet 18 的 conv2\_x 至 conv3\_x 变为 resnet 34 的 conv2\_x 至 conv3\_x。即如下：



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2 3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times 3, 64 \\ 3\times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3, 64 \\ 3\times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 64 \\ 1\times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 64 \\ 1\times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 64 \\ 1\times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times 3, 128 \\ 3\times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3, 128 \\ 3\times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128 \\ 1\times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128 \\ 1\times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128 \\ 1\times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times 3, 256 \\ 3\times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3, 256 \\ 3\times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256 \\ 1\times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256 \\ 1\times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256 \\ 1\times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times 3, 512 \\ 3\times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3, 512 \\ 3\times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512 \\ 1\times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512 \\ 1\times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512 \\ 1\times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

该曲线也描绘了更改结构后的模型性能。



其收敛的更快，并且最终精度略微好于其他几种结构。

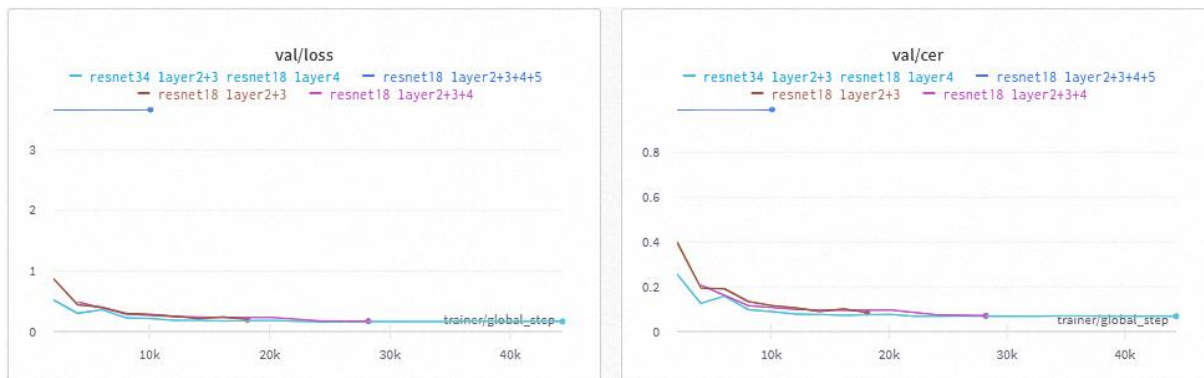
除此之外，我还测试了 number of head 这一参数，多头注意力的头数由于需要和 embedding 的维度匹配，因此可以选择的有 2, 4, 8, 16 等。在头数等于 8 的时候，模型很容易爆显存，并且性能没有什么明显变化。2 和 4 的性能也十分接近，分不出胜负。



通过查阅资料，我综合认为还是保留 4 的头数，以希冀在出现较为复杂的公式能保证模型的性能。

我们第一次测试上传的即为以上仅修改结构后的模型。排名为 13/32.

## 2. 模型的学习率和 batch size 最佳组合



如上面所提到的测试情况，模型显然在一定训练时间后便到达了训练瓶颈，此时 train loss 和 val loss 均不下降。

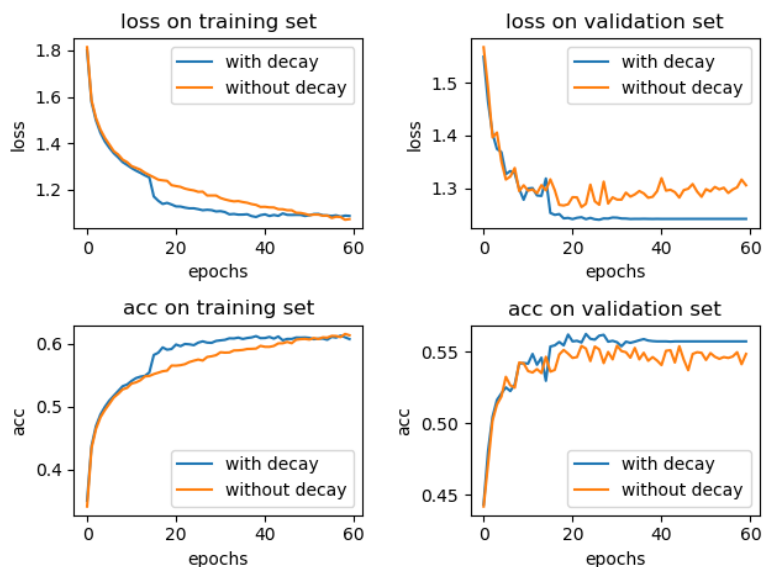
根据周志华的《机器学习》，机器学习的过程总是 - 欠拟合 -> 拟合 -> 过拟合，我推断这是处于接近拟合但可能未完全的情况。查阅资料，此时的对策应为减小 batch size 和学习率，其均在 config.yaml 文件中进行修改即可。

对于 batch size，原先为 64，现改为 16。

而对应的，学习率 lr 从原先的 0.001 降至 0.0004。在 lit\_resnet\_transformer.py 中，为了让模型的超参数更灵活，我采用了 Adamw 优化器，其是 Adam+L2 正则化的优化算法，保留了 Adam 动量的思想，自适应的学习率以调整权重矩阵，还通过 L2 正则化减少过拟合，整体优化后训练模型的速度更快。除此之外，我采用了多步下降（按需下降）的学习率指数衰减策略，将 scheduler 的 gamma 设为 0.5，milestone 则设为[5, 10, 15, 20, 25, 30, 35, 40, 45, 50...]. 原因是我观察发现 batch size 为 16 时一代的 global steps 数量已经能让其稳定至该学习率所能下降的极限范围，但为了增加探索的机会，以及考虑到 check validate dataset 的代数设为了 2，并不一定能准确保存到最好的模型，因此我给每个特定水平的学习率留下了 5 代的机会。

```
def configure_optimizers(self):
    optimizer = torch.optim.AdamW(self.model.parameters(), lr=self.lr, weight_decay=self.weight_decay)
    scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=self.milestones, gamma=self.gamma)
    return [optimizer], [scheduler]
```

关于采用了自适应学习率的优化器还采用了学习率衰减的策略是否会多余，我查阅了相关资料。资料显示，搭配使用效果反而更好。





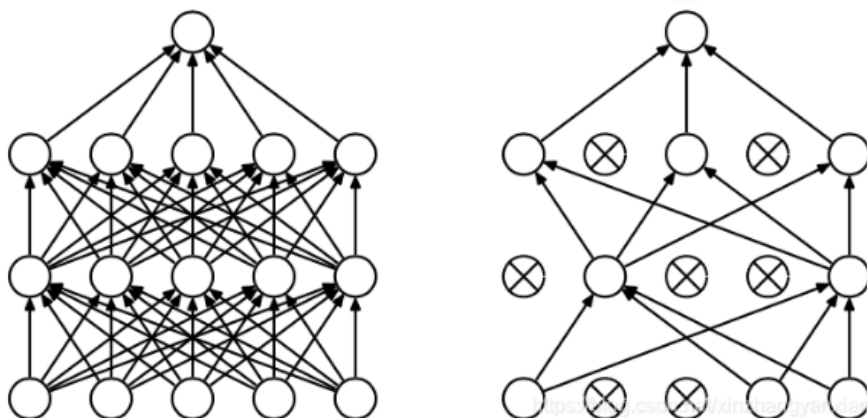
### 3. 模型的过拟合的解决方案



上述提到了机器学习的过程 - 欠拟合 -> 拟合 -> 过拟合。当模型遭遇过拟合的特征为 train loss 下降，val loss 不下降

可以看到上图绿色的曲线，val loss 随着训练反而在上升，符合过拟合的特征。

首先自然而然想到的措施是我们将 drop out 参数从 0.3 提升至 0.5。其是 transformer decoder layer 的一个参数。让更多的神经元随机的失活有助于减少神经元之间的依赖性，冗余性，迫使网络结构更灵活，更具有鲁棒性。



其次，我采用了数据增强的方法增加训练集图片特征的变化。一开始，我受博客启发采用的方法是图像融合。代码如下，主要是随机选取另一张图片并 resize 至相同的大小，进行数值上的随即比例加减以进行融合。

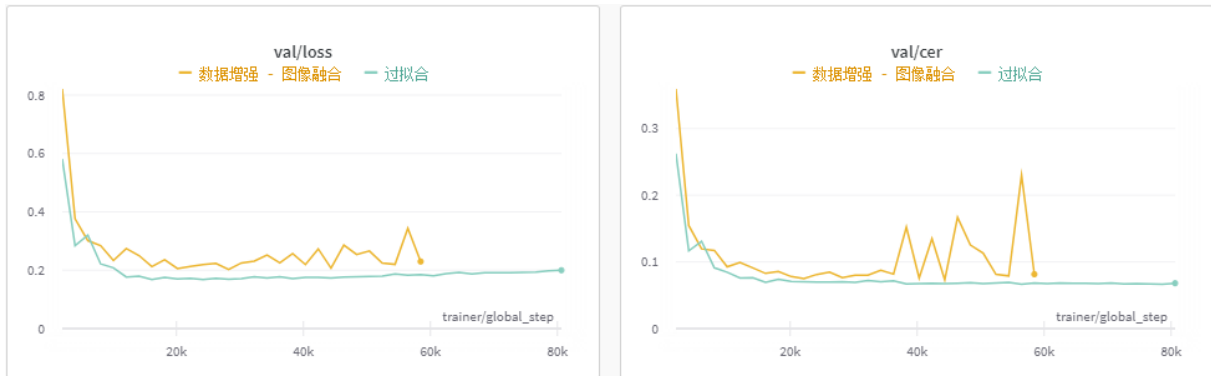
```

if self.transform is not None:
    image = self.transform(image=np.array(image))["image"]

# if self.train and idx>0 and idx%10==0:
#     mixup_idx=random.randint(0,len(self.image_filenames)-1)
#     mixup_image_filename, mixup_formula = self.image_filenames[mixup_idx], self.formulas[idx]
#     mixup_image_filepath = self.root_dir / mixup_image_filename
#     if mixup_image_filepath.is_file():
#         mixup_image = pil_loader(mixup_image_filepath, mode="L")
#     else:
#         # Returns a blank image if cannot find the image
#         mixup_image = Image.fromarray(np.full((64, 128), 255, dtype=np.uint8))
#         formula = []
# if self.transform is not None:
#     mixup_image = self.transform(image=np.array(mixup_image))["image"]
# if image.size()[2]-mixup_image.size()[2]>0:
#     wide_padding_h1 = int((image.size()[2] - mixup_image.size()[2]) / 2)
#     wide_padding_h2 = (image.size()[2] - mixup_image.size()[2]) - wide_padding_h1
#     pad = nn.ZeroPad2d((wide_padding_h1, wide_padding_h2, 0, 0))
#     mixup_image = pad(mixup_image)
# else:
#     wide_padding_h1=int((-image.size()[2]+mixup_image.size()[2])/2)
#     wide_padding_h2=(-image.size()[2]+mixup_image.size()[2])-wide_padding_h1
#     pad = nn.ZeroPad2d((wide_padding_h1, wide_padding_h2,0, 0))
#     image = pad(image)
# if image.size()[1]-mixup_image.size()[1]>0:
#     height_padding_v1=int((image.size()[1]-mixup_image.size()[1])/2)
#     height_padding_v2 = (image.size()[1]-mixup_image.size()[1])-height_padding_v1
#     pad = nn.ZeroPad2d((0,0,height_padding_v1,height_padding_v2))
#     mixup_image=pad(mixup_image)
# else:
#     height_padding_v1 = int((-image.size()[1] + mixup_image.size()[1]) / 2)
#     height_padding_v2 = (-image.size()[1] + mixup_image.size()[1]) - height_padding_v1
#     pad = nn.ZeroPad2d((0,0,height_padding_v1, height_padding_v2))
#     image = pad(image)
# alpha=0.5
# lam=np.random.beta(alpha, alpha)
# image=lam*image+(1-lam)*mixup_image
# formula=formula
return image, formula

```

但实际效果不佳，如上述所提到的以下这条曲线。在训练后期，图像融合会带来性能上很大的震荡，这引起了我的思考。



我猜想是图像融合可能并不适用于 image-to-latex 这个项目，毕竟手写公式与 imagenet 中的对图像识别分类存在一定差异性。手写公式具有序列性的特征，图像融合可能会丢失掉某些重要信息。为此，我采用了比较适用的高斯噪声、高斯模糊以及随机非等比例缩放等方式来进行数据增强，主要调用的库为 PIL 库。以下为部分代码：

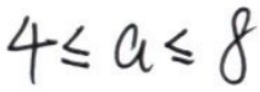
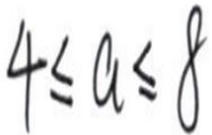

```

iw, ih = img.size
# 对图像进行缩放并且进行长和宽的扭曲
new_ar = iw / ih * random.uniform(1 - jitter, 1 + jitter) / random.uniform(1 - jitter, 1 + jitter)
scale = random.uniform(.15, 2.5)
if new_ar < 1:
    nh = int(scale * ih)
    nw = int(nh * new_ar)
else:
    nw = int(scale * iw)
    nh = int(nw / new_ar)
img = img.resize((nw, nh), Image.BICUBIC)
elif random.uniform(0,1)>0.5:
    img = img.filter(BLUR)
else:
    img=img.filter(GaussianBlur)
return img.convert(mode)

```

其中，随即非等比例缩放显然是一种不错的方法，因为不同人写字较大的区别即为字体的

高瘦不同，该方法能给图片带来更多的改变，也不会显著使图片变形难以辨认，可以很好地帮助模型的训练。

		
原始	高度增加	宽度增加

最后，一个不显著的方法为手动停止训练。过拟合常常是训练代数过多的情况出现的，并且很难完全避免，如果人能手动停止训练，也能较好地防止过拟合。

#### 4. 模型的训练加速

由于 batch size 被调整为 16，收敛到同等精度需要更多的训练时间

我们采用多进程技术为训练加速，number of worker 根据我们 CPU 的核数和内存大小调整为了 3，寻 batch 速度加快，下一轮迭代的 batch 在上一轮/上上一轮...迭代时已经加载好。对比发现模型每代训练的时间缩短了约 10s。

由于 batch size 为 16，显存负荷相对较少，因此 Pin\_memory 设为了 true，其会拷贝数据到 CUDA Pinned Memory，加快效率。

可以看到无论是哪个模型，其 GPU 的性能都有被充分的使用。



#### 5. 模型的预测加速

由于某些奇妙的缘由，服务器 load checkpoints 并用 model predict 时效率很低，平均每分钟只能预测 30 个公式。

我们采用多线程技术为预测加速，每个线程分到一部分的图片作为任务，最终平均每分钟能预测近 300 个公式（同样受到 CPU 核数的限制）

手搓的代码如下，比较简单：

```

lit_model = LitResNetTransformer.load_from_checkpoint("./artifacts/model.pt")
lit_model.freeze()
transform = ToTensorV2()
PATH="/root/autodl-tmp/image-to-latex-main/data/formula_images_processed/"
core=8
def funl(pid):
    global lit_model
    pid=eval(pid)
    result = []
    transform = ToTensorV2()
    with open("/root/autodl-tmp/image-to-latex-main/scripts/test_ids.txt", 'r') as f:
        txt = f.readlines()
        length=len(txt)
        inter_l=int(length/(core-1))*pid
        inter_r=min(length, int(length/(core-1))*(pid-1))
        for j in range(inter_l,inter_r):
            line = txt[j].replace('\n', '')
            path = PATH + line + ".png"
            if line == "65147" or line == "78755" or line == "84261" or line == "78755":
                result.append("-\n")
            else:
                image = Image.open(path).convert("L")
                image_tensor = transform(image.numpy.array(image))["image"]
                pred = lit_model.model.predict(image_tensor.unsqueeze(0).float())[0].numpy()
                decoded = lit_model.tokenizer.decode(list(pred))
                decoded_str = " ".join(decoded)
                result.append(decoded_str + "\n")
                print(decoded_str)
    with open("/root/autodl-tmp/image-to-latex-main/scripts/result"+str(pid)+".txt", 'w') as f:
        for i in result:
            f.write(i)

t0 = threading.Thread(target=funl, args=("0",))
t1 = threading.Thread(target=funl, args=("1",))
t2 = threading.Thread(target=funl, args=("2",))
t3 = threading.Thread(target=funl, args=("3",))
t4 = threading.Thread(target=funl, args=("4",))
t5 = threading.Thread(target=funl, args=("5",))
t6 = threading.Thread(target=funl, args=("6",))
t7 = threading.Thread(target=funl, args=("7",))
t0.setDaemon(True) #把子进程设置为守护进程，必须在start()之前设置
t1.setDaemon(True)
t2.setDaemon(True)
t3.setDaemon(True)
t4.setDaemon(True)
t5.setDaemon(True)
t6.setDaemon(True)
t7.setDaemon(True)

```

## 6. 模型的筛选和性能评估

在实际的对比中，我们发现模型提供的关于 val 的 loss 与 cer 指标都不能真正反映其在 validate dataset 的性能。例如我们第一次的测试结果无论在 val loss，train loss，val cer 上都优于排名前面的部分同学的结果。

因此我将模型一中内置的 BLUE-4、Exact Match Score、Edit Distance 指标迁移至模型二中，以方便我对模型的筛选。部分代码如下：

```

return file_names

lit_model = LitResNetTransformer.load_from_checkpoint("./artifacts/model.pt")
lit_model.freeze()
transform = ToTensorV2()
PATH="/root/autodl-tmp/image-to-latex-main/data/formula_images_processed/"
core=8

def funl(pid):
    global lit_model
    pid=eval(pid)
    formula_list=[]
    formula_pred_list=[]
    with open("/data/in2latex_formulas.norm.new.txt", 'r') as f:
        formula=f.readlines()
        with open("/data/in2latex_validate_filter.txt", 'r') as f:
            txt=f.readlines()
            length=len(txt)
            inter_l=int(length/(core-1))*pid
            inter_r=min(length, int(length/(core-1))*(pid-1))
            for i in range(inter_l,inter_r):
                line=txt[i]
                line=line.replace("\n", '')
                image_name=line.split(" ")[0]
                formula_idx=eval(line.split(" ")[1])
                formula=Formula[formula_idx].replace("\n", '')
                path = PATH + image_name
                if line == "65147.png" or line == "78755.png" or line == "84261.png" or line == "78755.png":
                    formula_pred=" "
                else:
                    image = Image.open(path).convert("L")
                    image_tensor = transform(image.numpy.array(image))["image"]
                    pred = lit_model.model.predict(image_tensor.unsqueeze(0).float())[0].numpy()
                    decoded = lit_model.tokenizer.decode(list(pred))
                    formula_pred = " ".join(decoded)
                    print(formula_pred)
                    formula_list.append(formula)
                    formula_pred_list.append(formula_pred)
    with open("/scripts/devtest"+str(pid)+".txt", 'w') as f:
        for i in range(0,len(formula_list)):
            f.write(formula_list[i]+"@"+formula_pred_list[i]+"\\n")
    # 子进程设置为守护进程，必须在start()之前设置

```

事实上我的决策非常正确，我数次与助教交流验证的结果，均十分符合我们的预期。

模型名称	BLUE-4	ExactMatchScore	EditDistance	Average Score	备注
第一次交的	87.966	65.811	89.571	81.116	resnet 34 layer2, 3 + resnet 18 layer4, lr 0.001 衰减一次 batchsize 64, dropout 0.3
第二次交的	90.744	72.922	92.056	85.24066667	resnet 34 layer2, 3 + resnet 18 layer4, lr 0.0004 多步衰减 batch size 16, dropout 0.5 图像增强
备选1	88.462	68.882	90.158	82.50066667	resnet 34 layer2, 3 + resnet 18 layer4, lr 0.0001 多步衰减 batch size 8, dropout 0.5 图像增强
备选2	89.499	71.324	91.572	84.13166667	resnet 50 layer2, 3, lr 0.0001 多步衰减 batch size 16, dropout 0.5 图像增强
备选3	89.233	68.153	91.613	82.99966667	resnet 34 layer2, 3, lr 0.0001 多步衰减 batch size 32, dropout 0.5 图像增强

由于第一次测试和第二次测试之间时间间隔较短，我将模型二的指南与用法交给了组员，共同跑数据与优化超参数。最终最好的结果以及第二好的结果实际不是在我个人电脑上运行的，但

是超参数的选取、模型的选择等均是我的指导。也非常感谢组员的辛勤付出。

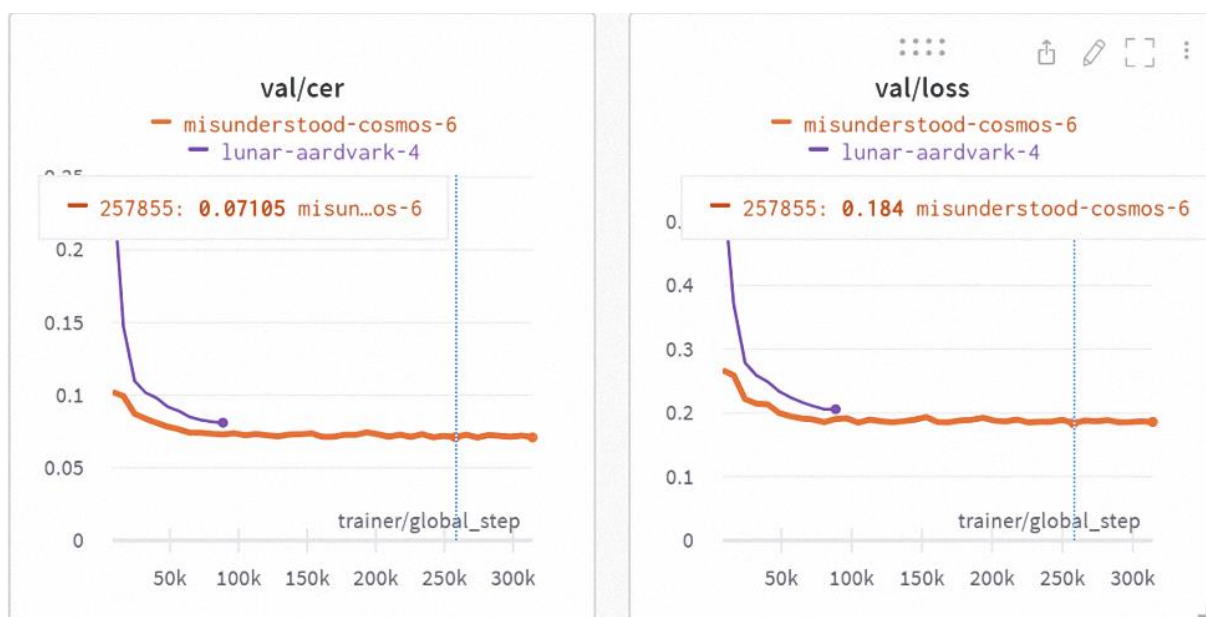
最终第二次测试结果名列 6/43，平均分数高达 81.3981，比第一次高了 5 分，排名上升 7 名，结果表明我们对模型的改进是正确的、有效的。

具体分数为：

BLUE - 4: 84.4714

Edit Distance: 88.7642

Exact Match Score: 70.9585



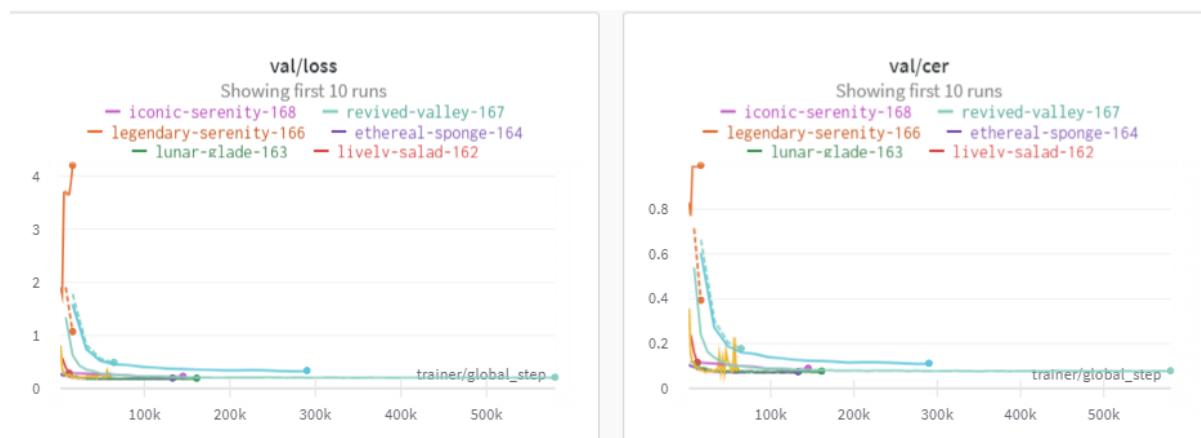
模型二的缺点

相比之前 batch size 为 64，该模型收敛的较为缓慢，在本次实验中较为注重精度而不惜计算资源。实际应用中要考虑精度和时效的平衡。

模型二的答辩以及 ppt 的制作



这是我跑过的所有模型：



## 小结

这次实验让我精进地了解了神经网络，从原理上，再到结构实现上，再到超参数调整上，再到数据质量优化上，夯实了理论基础，检验了自己的学习成果，学会了文献查阅，增强了代码实力。学以致用，在实践中对实际的开发或科研有所了解。

## 指导教师评语及成绩

评语：

成绩： 指导教师签名：  
 批阅日期：