

东北大学 毕业设计（论文）手册

起止时间： 2022 年 12 月 6 日-2023 年 5 月 31 日

学 院： 计算机科学与工程学院

专 业： 人工智能

学 号： 20195063

姓 名： 黄元通

指导教师： 栗伟

东北大学教务处印制
2022 年 12 月 1 号

毕业设计（论文）任务书

毕业设计（论文）题目：

基于 CLIP 模型的视频文本检索设计与实现

设计(论文)的基本内容：

查阅有关对比语言图像预训练、视频文本检索的文献资料，了解视频文本检索的相关技术及发展情况。

分析基于 CLIP 模型的视频文本检索需求，实现 CLIP4Clip 算法。设计并实现一个基于 CLIP/CLIP4Clip 模型的视频文本检索原型系统，即输入一段文本的描述，在某一段视频库中检索出与描述最相近的视频帧或者视频片段。

毕业设计（论文）专题部分：

题目： none

设计或论文专题的基本内容：

学生接受毕业设计（论文）题目日期

第 7 学期第 14 周

指导教师签字：



2022 年 12 月 1 号

工 作 记 录

查 阅 资 料 目 录

[1] Lei J, Li L, Zhou L, et al. Less is more: Clipbert for video-and-language learning via sparse sampling[C], Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021: 7331-7341.

[2] Devlin J, Chang M-W, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C]//Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, 2019: 4171-4186. doi: 10.18653/v1/N19-1423.

[3] Bain M, Nagrani A, Varol G, et al. Frozen in time: A joint video and image encoder for end-to-end retrieval[C], Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021: 1728-1738.

[4] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale[C]// 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.

[5] Bertasius G, Wang H, Torresani L. Is space-time attention all you need for video understanding?[C], ICML. 2021, 2(3): 4.

[6] Radford A, Kim J W, Hallacy C, et al. Learning transferable visual models from natural language supervision[C], International conference on machine learning. PMLR, 2021: 8748-8763.

[7] Luo H, Ji L, Zhong M, et al. CLIP4Clip: An empirical study of CLIP for end to end video clip retrieval and captioning[J]. Neurocomputing, 2022, 508: 293-304.

[8] Gao Z, Liu J, Chen S, et al. Clip2tv: An empirical study on transformer-based methods for video-text retrieval[J]. 2021.

[9] Fang H, Xiong P, Xu L, et al. Clip2video: Mastering video-text retrieval via image clip[J]. 2021.

[10] Wang J, Ge Y, Cai G, et al. Object-aware video-language pre-training for retrieval[C], Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022: 3313-3322.

[11] Wang Q, Zhang Y, Zheng Y, et al. Disentangled representation learning for text-video retrieval[J]. 2022.

[12] Jiang J, Min S, Kong W, et al. Tencent Text-Video Retrieval: Hierarchical Cross-Modal Interactions with Multi-Level Representations[J]. IEEE Access, 2022.

[13] Hounsby N, Giurciu A, Jastrzebski S, et al.

Parameter-efficient transfer learning for NLP[C], International Conference on Machine Learning. PMLR, 2019: 2790–2799.

[14] Pfeiffer J, Rücklé A, Poth C, et al. AdapterHub: A Framework for Adapting Transformers[C]// Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Online: Association for Computational Linguistics, 2020: 46–54. doi: 10.18653/v1/2020.emnlp-demos.7.

[15] Guo D, Rush A, Kim Y. Parameter-Efficient Transfer Learning with Diff Pruning[C]// Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). Online: Association for Computational Linguistics, 2021: 4884–4896. doi: 10.18653/v1/2021.acl-long.378.

[16] Li X L, Liang P. Prefix-Tuning: Optimizing Continuous Prompts for Generation[C]// Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). Online: Association for Computational Linguistics, 2021: 4582–4597. doi: 10.18653/v1/2021.acl-long.353.

[17] Lester B, Al-Rfou R, Constant N. The Power of Scale for Parameter-Efficient Prompt Tuning[C]// Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021: 3045–3059. doi: 10.18653/v1/2021.emnlp-main.243.

[18] Hu E J, Shen Y, Wallis P, et al. LoRA: Low-Rank Adaptation of Large Language Models[C]// The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25–29, 2022. OpenReview.net, 2022.

[19] Liu Y, Ott M, Goyal N, et al. Roberta: A robustly optimized bert pretraining approach[J]. 2019.

[20] He P, Liu X, Gao J, et al. DeBERTa: decoding-Enhanced Bert with Disentangled Attention[C]// 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3–7, 2021. OpenReview.net, 2021.

[21] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners[J]. OpenAI blog, 2019, 1(8): 9.

[22] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877–1901.

[23] Yang T, Zhu Y, Xie Y, et al. AIM: Adapting Image Models for Efficient Video Action Recognition[C]// The Eleventh International Conference on Learning Representations. 2023.

[24] Ramesh A, Pavlov M, Goh G, et al. Zero-shot text-to-image generation[C]//International Conference on Machine Learning. PMLR, 2021:

8821-8831.

[25] Karras T, Laine S, Aila T. A style-based generator architecture for generative adversarial networks[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019: 4401-4410.

[26] Patashnik O, Wu Z, Shechtman E, et al. Styleclip: Text-driven manipulation of stylegan imagery[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021: 2085-2094.

[27] Ashish V, Noam S, Niki P, et al. Attention is all you need[C], NeurIPS. 2017: pages 5998 - 6008.

[28] Xu J, Mei T, Yao T, et al. Msr-vtt: A large video description dataset for bridging video and language[C], Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 5288-5296.

[29] Gabeur V, Sun C, Alahari K, et al. Multi-modal transformer for video retrieval[C], Computer Vision - ECCV 2020: 16th European Conference, Glasgow, UK, August 23 - 28, 2020, Proceedings, Part IV 16. Springer International Publishing, 2020: 214-229.

[30] Yu Y, Kim J, Kim G. A joint sequence fusion model for video question answering and retrieval[C], Proceedings of the European Conference on Computer Vision (ECCV). 2018: 471-487.

[31] Chen D, Dolan W B. Collecting highly parallel data for paraphrase evaluation[C], Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies. 2011: 190-200.

[32] Kingma D P, Ba J. Adam: A Method for Stochastic Optimization[C]// 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. Eds. Bengio Y and LeCun Y. 2015.

学 生 工 作 记 录

毕业实习（调研）

单位：_____

完成任务情况：

学 生 工 作 记 录


设计工作与图纸绘制
(实验研究、论文工作) 完成日期: 5 月 19 号 (第 12 周)

设计说明书
(论 文) 完成日期: 5 月 31 号 (第 14 周)

专题部分完成日期:

毕业 设计
(论 文) 送交评阅人日期: 6 月 3 号 (第 14 周)

答辩日期: 06 月 08 日 (第 15 周)

指导教师签字: 

2023 年 6 月 15 日

上机 (实验) 地点:
东北大学

累计上机时数:
超过 550 小时

学 生 工 作 记 录

答 疑

内容提要:

1. 深入阅读了发表于 2021 的 CLIP 模型论文《Learning Transferable Visual Models From Natural Language Supervision》;
2. 深入理解 CLIP 的代码, 结合代码理清模型结构。

02 月 27 日 (第 1 周)

内容提要:

深入理解了 CLIP 的全部代码。

03 月 06 日 (第 2 周)

内容提要:

阅读和理解 CLIP4Clip 的代码, 但整个项目非常大, 代码非常的繁杂。

03 月 13 日 (第 3 周)

内容提要:

1. 完成了 CLIP4Clip 结构图的电子版;
2. 完成了特征存储向量存储模块。

03 月 20 日 (第 4 周)

内容提要:

在寒假期间学习了基本的 HTML、CSS 用于开发毕设的 Web 端界面, 现在继续学习 Django 来搭建后端。

03 月 27 日（第 5 周）

内容提要：

1. 使用 Django 完成了 Web 后端的搭建；
2. 构建了 MSR-VTT 的数据库，进行直接的文字查询用于结果对比；
3. 完善了 Web 界面，很多琐碎、复杂、耗费时间、又没有技术含量的 HTML、CSS 文件

04 月 03 日（第 6 周）

内容提要：

1. 广泛了解了 text prompt、visual prompt、PEFT (parameter efficient fine-tune)、关键帧抽取、细粒度特征交互等相关改进方法；
2. 实现了添加 Adapter（一种 PEFT 方式）的模型（2023 年的论文《AIM: adapting image models for efficient video action recognition》）；
3. 进行了一组训练实验。

04 月 12 日（第 7 周）

内容提要：

1. 整理了之前实验的数据，进行了一些绘图工作；
2. 写中期报告、中期答辩 PPT。

04 月 17 日（第 8 周）

内容提要：

1. 阅读了更多 CLIP 相关改进模型论文，整理了一下后续模型改进思路；
2. 模型改进实验。

04 月 24 日（第 9 周）

内容提要：

1. 实现了两种关键帧提取方案：平均选取、最大帧间差选取；
2. 实现了帧保存方案。

05 月 08 日（第 11 周）

内容提要：

1. 确定最终模型；
2. 进行最终所需的实验。

05 月 15 日（第 12 周）

内容提要：

1. 撰写毕业论文；
2. 修改毕业论文。

05 月 22 日（第 13 周）

指导教师签字：



2023 年 6 月 15 日

学 生 工 作 记 录

记 事

内容提要:

1. 毕设受实验条件限制，使用的是 ViT 系列 CLIP 中最小计算量最小的 ViT-B/32 版本、且不做在其他大视频数据集上的预训练，只进行在 MSRVTT 上的微调；meaP 效果只比使用 Transformer 做特征融合的效果低 3%左右，所以用 meaP。

2. JIT: Just In Time Compilation 即时编译（程序优化）

例如:

```
prog = re.compile(pattern) ←先编译
```

```
result = prog.match(string)
```

如果多次使用到某一个正则表达式，则建议先编译。

PyTorch 中就是将模型生成 TorchScript Module，模型是静态图。

3. cls: 用在 classmethod 函数里（代替 self）

cls 这个参数表示自身类，作用：调用类的属性，类的方法，实例化对象等；

在类函数里 cls() 就表示创建一个类

4. 模型是在半浮点下进行训练，在遇到指数操作时为保证基本的精度和防止溢出，需要转为浮点计算，例如在模型所有 LayerNorm 中，需要将 x 使用 torch.float32 计算，结果改回 x.dtype 传回；

5. CLIP 中激活函数使用的是其自实现的 QuickGeLU ($x * \text{sigmoid}(1.702 * x)$)，因为当时这样比 PyTorch 提供的 GeLU 快，但现在 PyTorch 已经改进了 GeLU。

6. 在 PyTorch 的多头自注意力层中，会将 (batch, seq, feature) 转置成 (seq, batch, feature)，以加快矩阵读取速度；同样，当 d_q、d_k、s_v 相等时合并成 in_proj_weight[d_model*3, d_model] 以加快矩阵读取速度。

7. CLIP 的 ViT 中，B/32 和 B/16 的 width 都是 768（网络其他地方都一样），这样做统一了模型 d，但 channel 从 14*14 减为了 7*7；

8. CLIP 中最后，在文字特征矩阵和图像特征矩阵相乘得到相似度矩阵时，乘了个可学习缩放倍数 logit_scale，其初值 1/0.07 没有特别含义，可能单纯是这样就效果好；

9. CLIP 中 conv、Linear、自注意除了 Wc、Transformer 最后的矩阵、ViT 最后的矩阵都是半精度以加快训练速度。

内容提要：

1. @lru_cache()

@cache 是一项优化技术，把函数的结果保存起来，避免传入相同的参数时重复计算；

@lru_cache 是最近最少使用置换算法，默认最多保存 maxsize=128 个结果

2. CLIP 的 tokenizer:

a) 每个 Unicode 字符，UTF-8 用 1~4 个字节编码（1Bit=8bit）；

b) 为避免、减小字典大小，将未见过的 UTF-8 字符拆碎成字节，{字节值: Unicode 字符} 长度 256，里面是[a A 1 , 乱码]等基本字符；同时避免映射到 bpe 代码所依赖的空白/控制字符；

c) 格式化字符串的方法使用了：

i. ftfy.fix_text -> html.unescape -> strip: 把损坏的 Unicode 还原成正确的字符、把一些 HTML 里的编码替换为 Unicode 字符；

ii. 去除空白字符 -> strip, re 正则式里\s 是空白字符，如\t\n\r\f\v 等；

d) encode: 一句话->[单词]->[分子词（以字节划分）]->[index(分子词)]；

e) decoder: [index(分子词)]->[分子词]->[单字节值]->[UTF-8 字符]；

f) 构造出的字典长度为 49408，包括：

i. [256 个 Unicode 字符, 256 个 Unicode 字符]，表示分子词位于单词末尾；

ii. [48894 个分子词]，是保存在 bpe_simple_vocab_16e6.txt.gz 里的，事先统计好的、按概率排序的、分子词

iii. 2 个特殊字符：['<|startoftext|>', '<|endoftext|>']

3. Tokenizer 里的 BPE 算法：

a) 功能：将未见过的词拆成分子词。例如:输入 aword，输出 a word；

b) 把 word 拆成('w', 'o', 'r', 'd') ->

生成 set(('w', 'o'), ('a', 'w'), ('o', 'r'), ('r', 'd')), 无序 ->

将 token 拆分成数量尽可能少的分子词（贪心算法）：将 pairs 按照 bpe_ranks

中的顺序排序，找到最常用的那个 pair（在文件中排在最前面的），找不到的 pair 返回无穷大 float('inf') 以免被选上 →

c) 组合不在 bpe 词表中，pairs 不能再合并了，循环结束；

03 月 06 日（第 2 周）

内容提要：

1. Python argparse 库：

命令行选项、参数和子命令解析器，可以用来方便地读取命令行参数：

a) 首先使用 ArgumentParser() 创建解析器，然后 add_argument() 添加需要解析的参数，最后 parse_args() 解析参数；

b) add_argument() 中的参数：

i. name_or_flags:

--name: 可选参数，赋值时只能使用名字查找：--name 值；

name: 位置参数，赋值时不可按名字查找，只能按输入参数顺序赋值；

ii. action:

store_const: 不可对 name 传入值，若命令行输入 --name，那么 name 中保存 const 关键字指定的值；

store_true、store_false: 不可对 name 传入值，若命令行输入 --name，那么 name 中保存 True、False；

append: 将同一参数的不同值保存在一个 list 中；

iii. default: 指定参数默认值，默认为 None；

iv. type: int、float、open(文件) 等，默认为 str；

v. choices: 将命令行参数的可输入值使用列表进行限定；

vi. required: 必须输入参数；

vii. help: 当用户请求帮助时（一般是通过在命令行中使用 -h 或 --help 的方式），这些 help 描述将随每个参数一同显示。

2. 主要需要调整的参数：

- a) train_csv: 训练集 CSV 路径;
- b) val_csv: 验证集 CSV 路径;
- c) data_path: 数据集的文字描述路径;
- d) features_path: 数据集的视频;
- e) num_thread_reader: PyTorch 里 Dataloader 中子线程数;
- f) feature_framerate: 单个视频片段每秒采样帧数;
- g) pretrained_clip_name: 选用的 CLIP 模型名;

3. 手绘了史上最详细的 CLIP 模型的结构、流程、每一步的特征向量形状变化图:

记 事

内容提要:

1. 毕设受实验条件限制,使用的是 ViT 系列 CLIP 中最小计算量最小的 ViT-B/32 版本、且不做在其他大视频数据集上的预训练,只进行在 MSRVT 上的微调;meaP 效果只比使用 Transformer 做特征融合的效果低 3%左右,所以用 meaP。
2. JIT: Just In Time Compilation 即时编译(程序优化)
例如:

```
prog = re.compile(pattern) ←先编译
result = prog.match(string)
```

如果多次使用到某一个正则表达式,则建议先编译。

PyTorch 中就是将模型生成 TorchScript Module,模型是静态图。
3. cls: 用在 classmethod 函数里(代替 self)
cls 这个参数表示自身类,作用:调用类的属性,类的方法,实例化对象等;
在类函数里 cls()就表示创建一个类
4. 模型是在半浮点下进行训练,在遇到指数操作时为保证基本的精度和防止溢出,需要转为浮点计算,例如在模型所有 LayerNorm 中,需要将 x 使用 torch.float32 计算,结果改回 x.dtype 传回;
5. CLIP 中激活函数使用的是其自实现的 QuickGeLU ($x * \text{sigmoid}(1.702 * x)$),因为当时这样比 PyTorch 提供的 GeLU 快,但现在 PyTorch 已经改进了 GeLU。
6. 在 PyTorch 的多头自注意力层中,会将 (batch, seq, feature)转置成(seq, batch, feature),以加快矩阵读取速度;同样,当 d_q、d_k、s_v 相等时合并成 in_proj_weight[d_model*3,d_model]以加快矩阵读取速度。
7. CLIP 的 ViT 中, B/32 和 B/16 的 width 都是 768(网络其他地方都一样),这样做统

一了模型 d, 但 channel 从 14*14 减为了 7*7;

8. CLIP 中最后, 在文字特征矩阵和图像特征矩阵相乘得到相似度矩阵时, 乘了个可学习缩放倍数 `logit_scale`, 其初值 `1/0.07` 没有特别含义, 可能单纯是这样就效果好;
9. CLIP 中 `conv`、`Linear`、自注意除了 `Wc`、`Transformer` 最后的矩阵、`ViT` 最后的矩阵都是半精度以加快训练速度。

02 月 27 日 (第 1 周)

内容提要:

1. `@lru_cache()`

`@cache` 是一项优化技术, 把函数的结果保存起来, 避免传入相同的参数时重复计算;
`@lru_cache` 是最近最少使用置换算法, 默认最多保存 `maxsize=128` 个结果

2. CLIP 的 tokenizer:

- a) 每个 Unicode 字符, UTF-8 用 1~4 个字节编码 (1Bit=8bit);
- b) 为避免 `<unk>`、减小字典大小, 将未见过的 UTF-8 字符拆碎成字节, {字节值: Unicode 字符} 长度 256, 里面是 [a A 1, 乱码] 等基本字符; 同时避免映射到 `bpe` 代码所依赖的空白/控制字符;
- c) 格式化字符串的方法使用了:
 - i. `ftfy.fix_text -> html.unescape -> strip`: 把损坏的 Unicode 还原成正确的字符、把一些 HTML 里的编码替换为 Unicode 字符;
 - ii. 去除空白字符 `-> strip`, `re` 正则式里 `\s` 是空白字符, 如 `\t\n\r\f\v` 等;
- d) `encode`: 一句话 `->` [单词] `->` [分子词 (以字节划分)] `->` [index(分子词)];
- e) `decoder`: [index(分子词)] `->` [分子词] `->` [单字节值] `->` [UTF-8 字符];
- f) 构造出的字典长度为 49408, 包括:
 - i. [256 个 Unicode 字符, 256 个 Unicode 字符 `</w>`], `</w>` 表示分子词位于单词末尾;
 - ii. [48894 个分子词], 是保存在 `bpe_simple_vocab_16e6.txt.gz` 里的, 事先统计好的、按概率排序的、分子词
 - iii. 2 个特殊字符: [`<|startoftext|>`, `<|endoftext|>`]

3. Tokenizer 里的 BPE 算法:

- a) 功能: 将未见过的词拆成分子词。例如: 输入 `aword`, 输出 `a word`;
- b) 把 `word` 拆成 (`'w','o','r','d</w>'`) `->`
生成 `set(('w','o'),('a','w'),('o','r'),('r','d</w>'))`, 无序 `->`
将 `token` 拆分成数量尽可能少的分子词 (贪心算法): 将 `pairs` 按照 `bpe_ranks` 中的顺序排序, 找到最常用的那个 `pair` (在文件中排在最前面的), 找不到的 `pair` 返回无穷大 `float('inf')` 以免被选上 `->`

c) 组合不在 `bpe` 词表中, `pairs` 不能再合并了, 循环结束;

03 月 06 日 (第 2 周)

内容提要:

1. Python argparse 库:

命令行选项、参数和子命令解析器, 可以用来方便地读取命令行参数:

a) 首先使用 `ArgumentParser()` 创建解析器, 然后 `add_argument()` 添加需要解析的参数, 最后 `parse_args()` 解析参数;

b) `add_argument()` 中的参数:

i. `name_or_flags`:

`--name`: 可选参数, 赋值时只能使用名字查找: `--name 值`;

`name`: 位置参数, 赋值时不可按名字查找, 只能按输入参数顺序赋值;

ii. `action`:

`store_const`: 不可对 `name` 传入值, 若命令行输入 `--name`, 那么 `name` 中保存 `const` 关键字指定的值;

`store_true`、`store_false`: 不可对 `name` 传入值, 若命令行输入 `--name`, 那么 `name` 中保存 `True`、`False`;

`append`: 将同一参数的不同值保存在一个 `list` 中;

iii. `default`: 指定参数默认值, 默认为 `None`;

iv. `type`: `int`、`float`、`open`(文件)等, 默认为 `str`;

v. `choices`: 将命令行参数的可输入值使用列表进行限定;

vi. `required`: 必须输入参数;

vii. `help`: 当用户请求帮助时 (一般是通过在命令行中使用 `-h` 或 `--help` 的方式), 这些 `help` 描述将随每个参数一同显示。

2. 主要需要调整的参数:

a) `train_csv`: 训练集 CSV 路径;

b) `val_csv`: 验证集 CSV 路径;

c) `data_path`: 数据集的文字描述路径;

d) `features_path`: 数据集的视频;

e) `num_thread_reader`: PyTorch 里 `Dataloader` 中子线程数;

f) `feature_framerate`: 单个视频片段每秒采样帧数;

g) `pretrained_clip_name`: 选用的 CLIP 模型名;

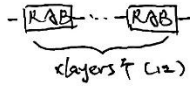
3. 手绘了 CLIP 模型的结构、流程、每一步的特征向量形状变化图:

Residual Attention Block (preNorm, head=8)

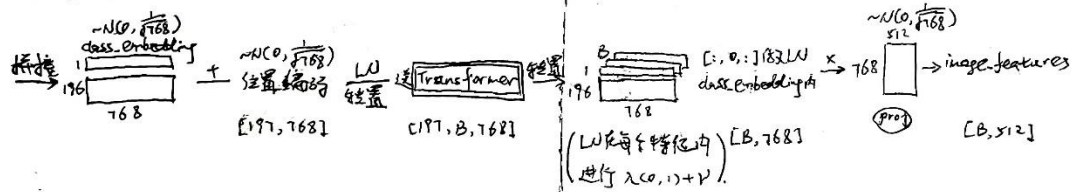
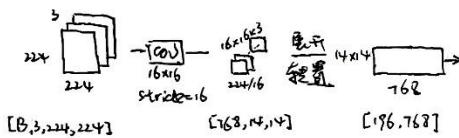


QuickGelu = $x \cdot \text{sigmoid}(1.702 \cdot x)$

Transformer

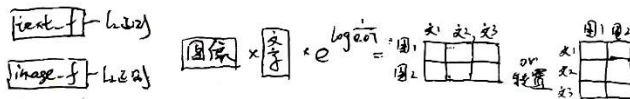


ViT (patch_size=16) (patch_size=32时, channels=7*7+1=49+1)

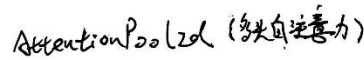


(在CLIP4Clip里被挪到了ViT外面)

CLIP:



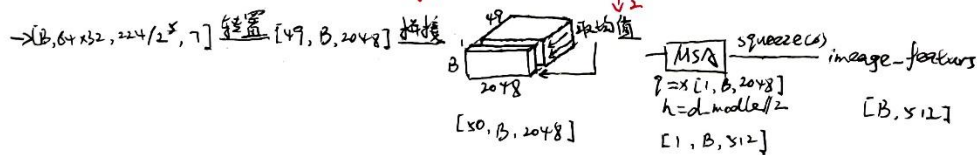
BottleNeck. \boxtimes (inplanes, planes, stride)
as pool



$$\boxed{z_1} \boxed{z_2} \dots \boxed{z_n} \times \boxed{w_c} = \boxed{z}$$

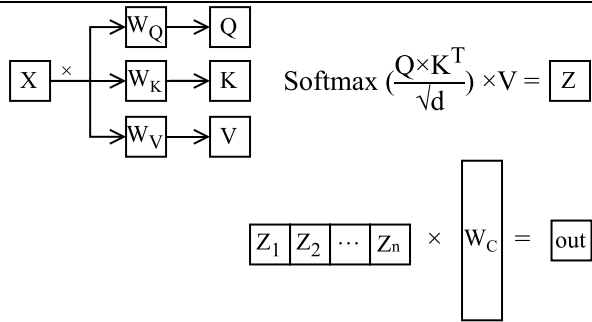
Diagram illustrating a sequence of operations (likely a neural network layer structure) involving convolutions and pooling:

Input: $[8, 3]$ (with $248, 244$ below it) is processed by a **conv** operation (kernel size 3, stride 2) to produce a 6x6 feature map. This is followed by a **relu** operation. The resulting 6x6 feature map is then processed by another **conv** operation (kernel size 3, stride 2) to produce a 3x3 feature map. This is followed by a **relu** operation. The resulting 3x3 feature map is then processed by a third **conv** operation (kernel size 3, stride 2) to produce a 1x1 feature map. This is followed by a **relu** operation. Finally, an **avg pool** operation is applied to the resulting 1x1 feature map.

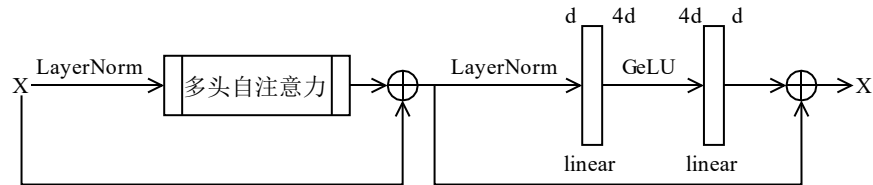


- R@1: 42.2 - R@5: 70.2 - R@10: 80.9 - Median R: 2.0 - Mean R: 15.3

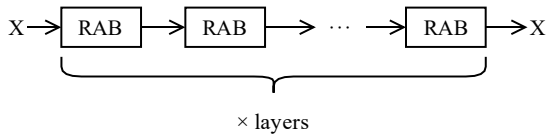
1) 多头自注意力:



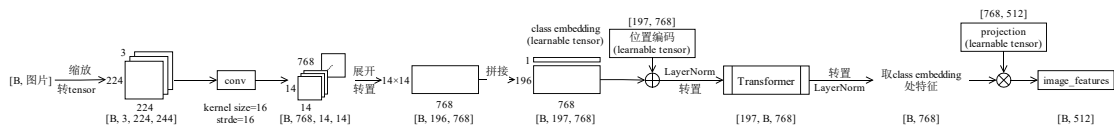
2) ResidualAttentionBlock:



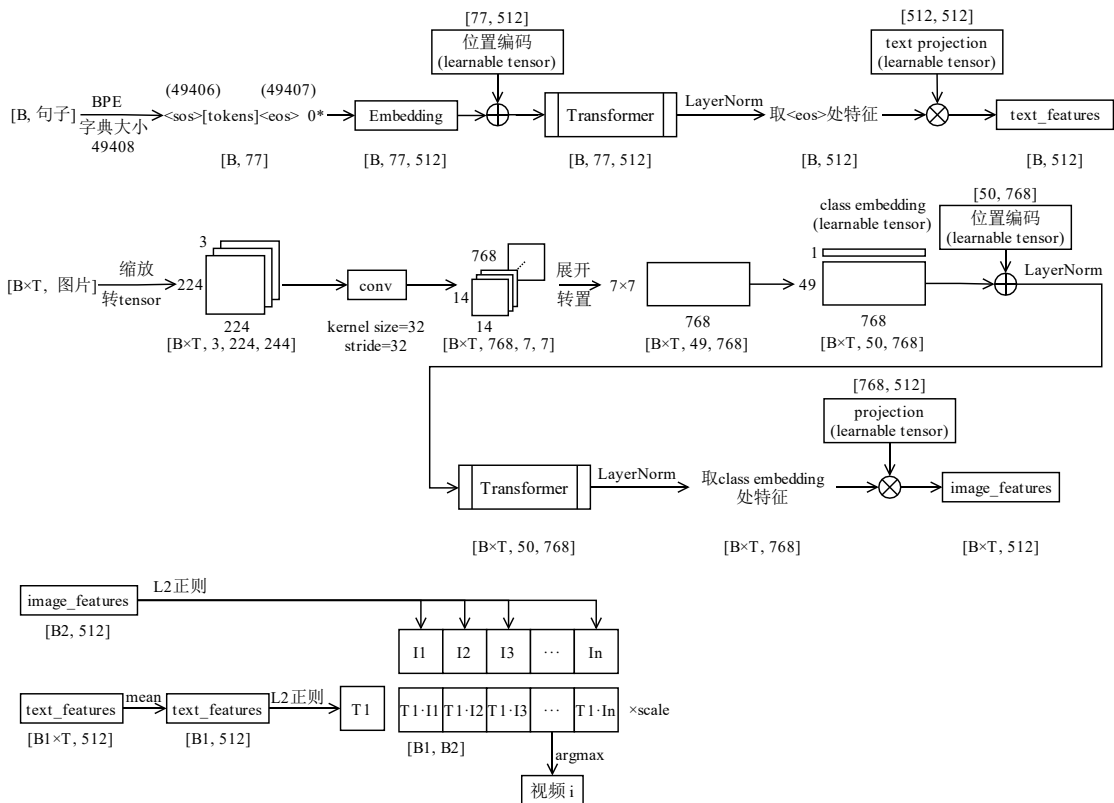
3) Transformer:



4) ViT:



5) CLIP4Clip:



2. 特征存储向量存储模块 DataBase:

1) 根据不同数据库名使用相应文件分开存储;

2) 数据库名称关键字: test、raw、train、raw、text;

3) Notes:

a) 数据库中所有数据使用 `numpy.ndarray`(数字型)存储, `load` 也是返回相应 `array` 矩阵, `store` 前请将数据都转为 `array`;

b) 数据集操作目前为 `pandas`, 所有 `DataFrame` 相关数据、操作视为对外界不可见, 抽象为 `array` 的存储和访问;

c) `text_feature`、`video_features_raw` 已经过 L2 正则; `video_features` 已经过 `Tool.meanP_video_features`

d) 数据库名称关键字对应数据列:

`MSRVTT_test`: [`video_id`, `text_feature`, `video_features`]

`MSRVTT_test_raw`: [`video_id`, `text_feature`, `video_features_raw`]

`MSRVTT_train_video`: [`video_id`, `video_features`]

`MSRVTT_train_video_raw`: [`video_id`, `video_features_raw`]

`MSRVTT_text`: [`video_id`, `text_feature`]

4) 主要包括以下 4 个对外的函数接口:

a) `store(cls, database_name: str, *args):`

保存 `features` 到相应 `pickle` 文件

:param `database_name`: `DataBase`.数据库名

:param `args`: 要保存的特征, 按照顺序: `video_id`(必选)、`text_feature`、`video_features`、`video_features_raw`, 只能 2 个或 3 个

b) `load(cls, database_name: str):`

从保存的 `pickle` 文件中加载相应 `database`

:return: `list(numpy.ndarray)`

c) `name2index(content):`

考虑到以后若加入其他数据集进入程序, 为了每个数据集内所有视频(特征)都不会出现重名, 所以特此设定此函数用于将文件名与在 `DataBase` 中序号互相转换, 目前只有 `MSRVTT` 的情况下, 转换标准就是: 'video1' -> 1, 1 -> 'video1'。

d) `get_top_n_info(cls, sim_matrix: numpy.ndarray, video_id: numpy.ndarray, top_n: int):`

用于主程序使用用户输入文字计算出相似矩阵后、转换为相应视频编号, 考虑到所有特征保存、加载功能都集成在此模块中, 所以也将此功能归入 `DataBase` 内;

返回一个文字-视频相似矩阵中, 相似度最大 `top_n` 个视频的: [(`video_id`、`sentence`、相似度)]。

3. 关于 DataBase 要用来进行操作、用什么文件形式保存、以什么数据格式保存等问题，做了以下几组实验，最终发现使用 Pandas 用 DataFrame 操作、用 pickle 存储 numpy.ndarray 具有最快的速度，因此才选定最终 DataBase 的数据保存方式。

- 1) pickle 保存 torch.tensor;
- 2) torch.tensor 转为 numpy.array 后，再 pickle 保存;
- 3) Pandas DataFrame 中存储 torch.tensor，然后 to_pickle 保存;
- 4) torch.tensor 转为 numpy.array 后，Pandas DataFrame 中存储 numpy.array，然后 to_pickle 保存。

1) 实验设置:

a) 模拟数据量:

video_id: [10000,], list[int]

text_features: [10000, 1, 512], torch.tensor(torch.float32)

video_features: [10000, 12, 512], torch.tensor(torch.float32)

= 共计 266.28 million 字节

b) 进行的操作:

全部数据进行一次保存、一次加载、转回 torch.tensor 后、进行 1 次切片、2 次索引、一次[512,]@[512,]矩阵乘法。

2) 实验结果:

	时间 (s)	文件大小 (MB)	备注
方式 1	1.906	253	正常运行
方式 2	0.828	253	正常运行
方式 3	∞	$\geq 25.1\text{GB}$	运行, MemoryError 终止
方式 4	0.718	254	正常运行

4. 另外在网上看到了 Pandas 保存为 feather 文件可以进一步提高数据存储、读取速度，但实验中发现 feather 并不支持保存矩阵。
5. 另外还简单了解了向量数据库 FAISS，是 FaceBook 发布的一个可用于快速向量检索的框架，但老师和我都认为当前 10 万以下的量级上，直接用所有向量相乘或遍历的时间开销仍然很低，没有使用向量检索框架的必要性。

03 月 20 日（第 4 周）

内容提要:

1. 看的是一位技术博主 Dennis Ivanov 的视频进行学习，使用的的项目是一个类似 Twitter 的博主自己设计教学小项目:

<https://github.com/divanov11/StudyBud>

2. 另外还查阅了很多 Django 官方手册: <https://docs.djangoproject.com/zh-hans/4.1/>

3. 静态资源管理

Web 后端的一项关键任务是静态资源管理, Django 中, 图片、CSS 等文件都属于静态文件, 通过在配置文件 (setting.py) 中配置 STATICFILES_DIRS 和 STATIC_URL 实现加载。

例如在项目根目录下创建文件夹 “image” 用于存放图片, 那么就应该将 image 的路径添加到 STATICFILES_DIRS 中, 并可设置命名空间 (prefix) 为 “命名空间 1”, 用命名空间来区分不同文件夹下的同名文件。若有文件 “image/子文件夹/图片.jpg”, 那么在代码中通过 “{% static '命名空间 1/子文件夹/图片.jpg' %}” 或是 “{% get_static_prefix %}命名空间 1/子文件夹/图片.jpg” 访问。

而 STATIC_URL 是静态文件在网页中显示的链接的前缀, 通常设为 “static/”。例如上例中的 “图片.jpg”, 那么这时其在网页中所显示的完整链接就是 “http://网页地址/static/命名空间 1/子文件夹/图片.jpg”。

在本程序中, 主要使用到 2 个静态资源文件夹, 分别是视频根目录和视频封面根目录。视频根目录下存放的是 MSR-VTT 数据集的原始视频, 用于在视频播放页面进行视频播放, 命名空间使用 videos; 视频封面根目录下存放的是所有视频的封面, 用于在搜索结果展示界面进行展示, 命名空间使用 images。

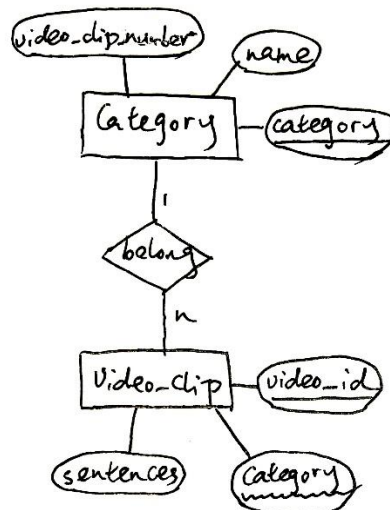
因为 MSR-VTT 数据集的视频文件命名规则是 “video 视频编号.mp4”, 所以可以方便地设置视频对应封面文件名为 “视频编号.jpg”, 这样, 只需要给定视频编号, 通过 “{% get_static_prefix %}videos/video 视频编号.mp4” 和 “{% get_static_prefix %}images/视频编号.jpg” 便可访问到对应的视频和视频封面。

03 月 27 日 (第 5 周)

内容提要:

1. 数据库:

使用 Python 和 Django 自带的轻量级 SQLite, 保存了测试集中视频编号与所有文字描述等信息, 数据库 E-R 模型如下:



2. Web 后端:

需求:

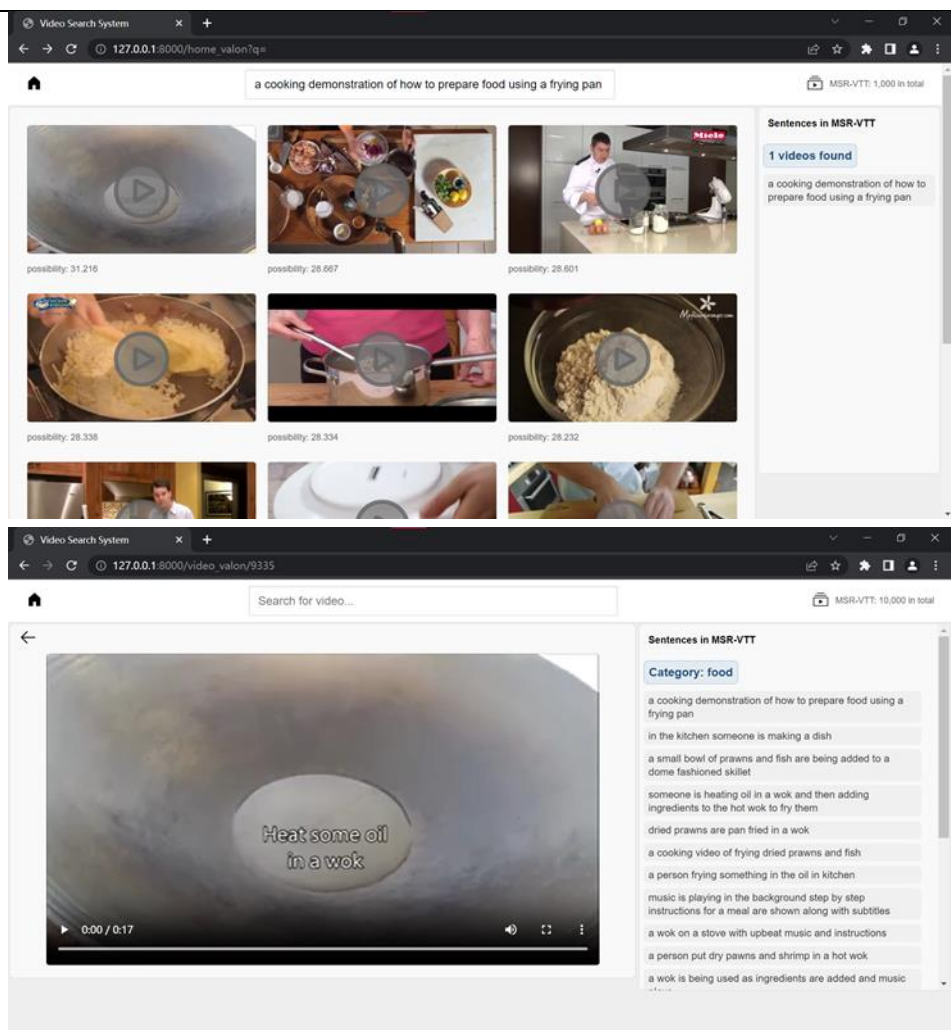
- 1) 用户输入文字, 需要使用模型快速地从 MSR-VTT 测试集中检索到最佳的 top-k 个视频, 并展示各视频预览, 用户点击后可进入该视频播放页面;
- 2) 用户可选择开启文字检索, 通过数据库文字匹配返回搜索结果, 以作为与模型视频检索结果的对比。

实现:

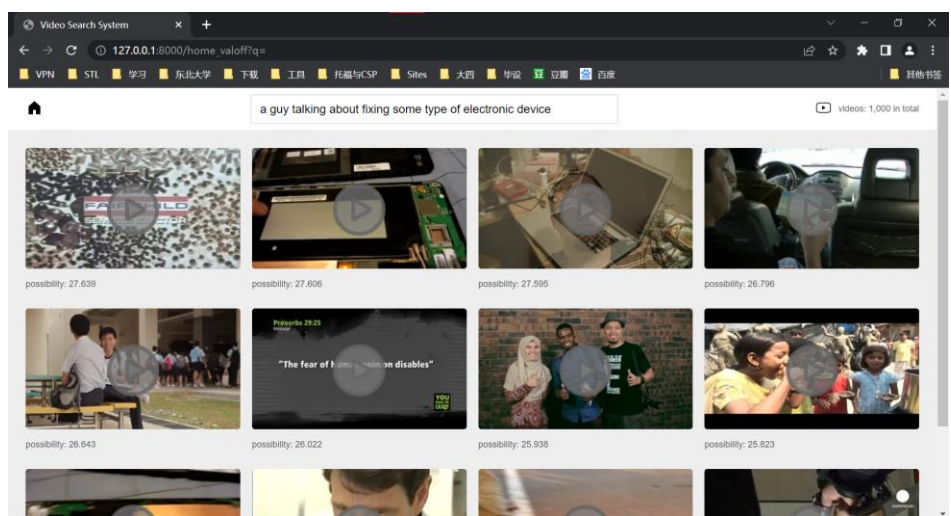
- 1) 充分利用模型的双塔结构特性, 将 MSR-VTT 测试集视频全部预先抽取特征, 并保存在自实现的“特征数据库 DataBase”中 (使用 Pandas 将矩阵转为 numpy, 并以 pickle 文件保存), 1 万条特征大小为 19MB, 数据加载时长为 0.1 秒数量级;
- 3) 用户输入文字后通过模型文字编码器得到文字特征, 然后和保存好的视频特征进行一次矩阵乘法即可得到结果, 可直接在 CPU 计算, 1 万条视频处理时长为 0.1 秒数量级;
- 4) 在 Django 中, 界面划分为“模型检索结果”、“数据库查找结果”2 栏, 使用数据库 icontains 查找在所有文字描述中匹配用户输入, 将结果显示在“数据库查找结果”栏中以供用户对比参考。

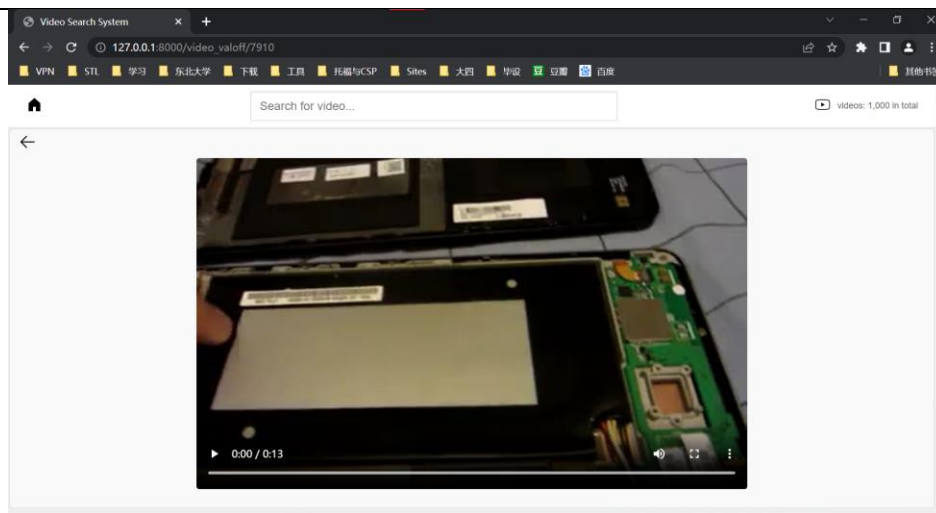
3. Web 界面:

界面设计以寒假里学习的 YouTube 界面、第 5 周 Django 学习项目的界面为基础:

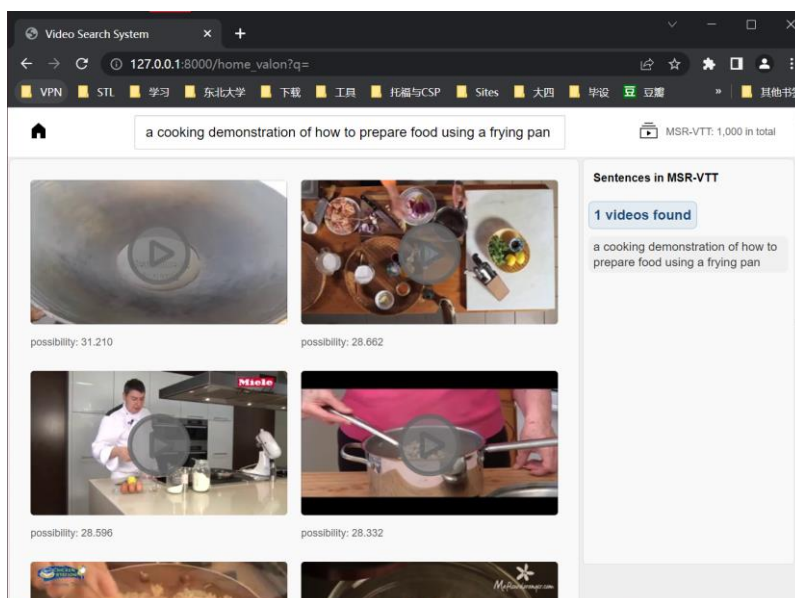


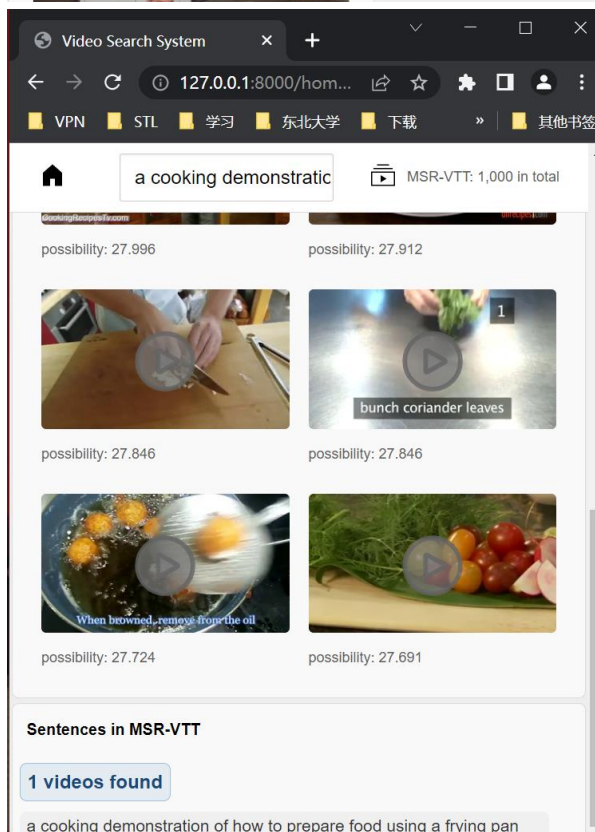
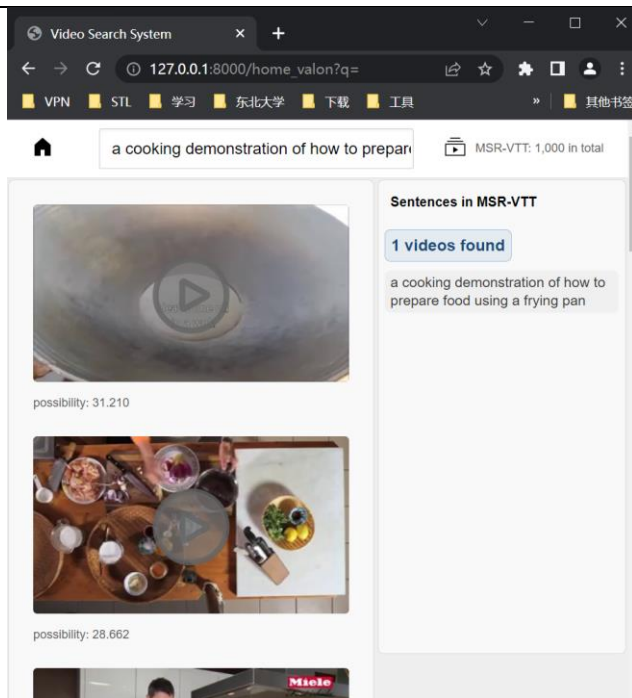
考虑到实际应用中，该视频检索模型的应用场景下视频应该没有文字描述，所以想要将“开启文字检索”设计为用户的可选项，只是在这些数据集内用作参考。将“开启文字检索”称谓“evaluation on”、反正则称为“evaluation off”，用户通过点击右上角按钮进行状态切换：





4. 另外，还做了很多的 Web 界面布局自适应工作，以适应不同的窗口大小带来更好的浏览体验，但目前还未在手机端测试过效果：





04 月 03 日（第 6 周）

内容提要:

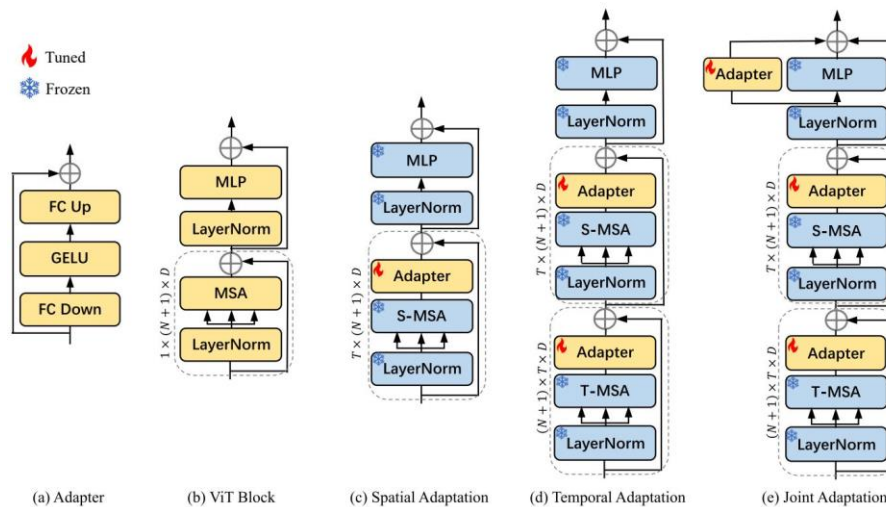
1. 模型设计:

AIM 是 2023 动作识别的一篇论文，其思想是冻结整个网络、只训练在 Transformer

层自注意力后添加的 Adapter 以降低训练开销，以极少的参数加强模型空间、时间信息学习能力；

Adapter 结构非常简单：FC 四倍下采样→激活层→FC 四倍上采样，并用残差连接，以 CLIP4Clip 中 ViT 为例，768 的特征长度，Adapter 只含有约 $768 \times 192 \times 2$ 个参数；添加的总参数量为 1430 万，而原 CLIP4Clip 总参数量为 1 亿 5000 万，训练参数量较 full fine-tune 减少至 9.5%；

对于 $[\text{batch size} \times \text{time}, \text{channel}, D]$ 的特征，自注意力操作在 channel 间交互各 D 以计算注意力；因此，时间 Adapter 将矩阵在进入自注意力前转置为 $[\text{batch size} \times \text{channel}, \text{time}, D]$ 从而让各 D 在 time 间交互；空间 Adapter 将矩阵在进入自注意力前转置回 $[\text{batch size} \times \text{time}, \text{channel}, d]$ （两处的自注意力权重使用同一份）。



2. 模型实现：

代码在 Improve.py 中，我的出发点是尽量保持原 CLIP4Clip 模型的完整性，所以我使用的是 partial 函数重写原模型的 forward 函数（我找了很久网上资料，但没有发现有什么其他更好方法进行此函数的重写，因为 PyTorch 里的 forward 是会被 Module 基类里的一个方法自动调用的，所以比较特殊，要重写比较麻烦）。

3. 实验设置：

Adapter 模块同样使用半浮点以和网络整体基调统一。主要是为了测试这种方法是否有效，所以我只添加了 Temporal Adaptation、且训练集使用的是 MSRVT-7K。

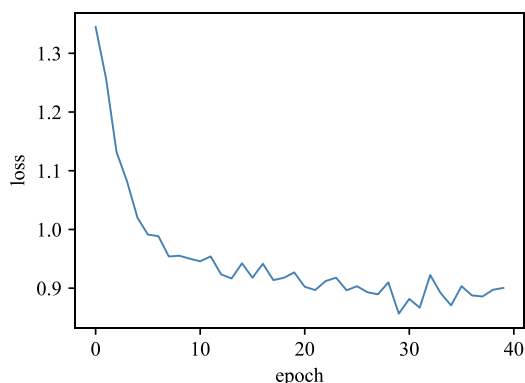
并且每视频 20 个描述的使用方式是：每个 epoch 中只随机选取一个，不像 CLIP4Clip full fine-tune 时每视频 20 个描述在每个 epoch 中全部参与，所以这种情况下 20 个 epoch 的数据量之和与 1 个全部参与 epoch 的数据量相等。

实验中发现就算冻结了整个 CLIP4Clip，但仍然是学习率敏感的，AIM 论文中使用的是 $3e-4$ ，但实验了 $1e-4$ 、 $3e-4$ 均会跑飞，损失一直不降然后在 epoch 2 时突然发生上溢变 nan；使用 $1e-7$ （CLIP4Clip full fine-tune 时 CLIP 内参数学习率为 $1e-7$ ，

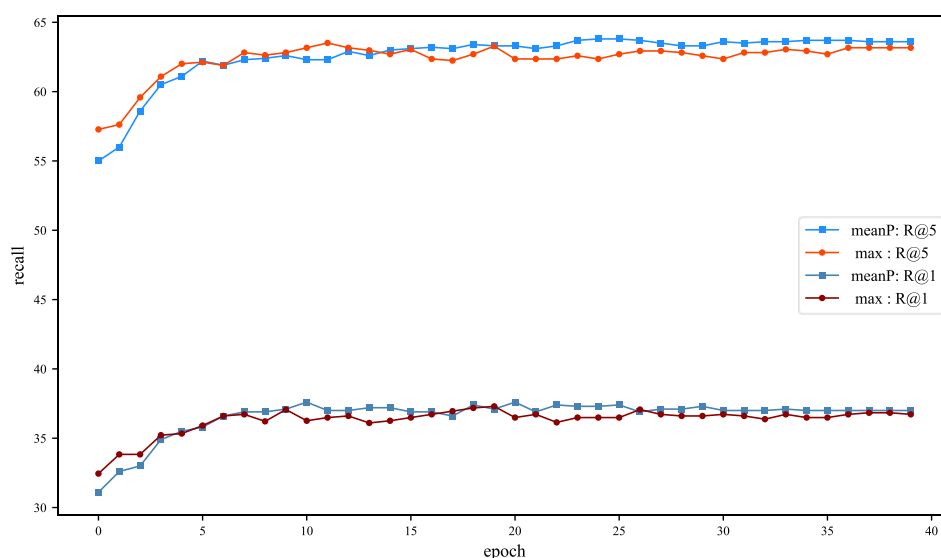
CLIP 外参数学习率为 $1e-4$) 时在 epoch 10 时基本停止收敛; 之后还尝试了 $1e-6$ 但没有 $1e-7$ 的效果佳。

4. 实验结果:

损失变化:



在测试集上召回率变化:



5. 思考:

AIM 是作者应用于视频动作识别领域的模型, 模型结构是 CLIP 的 ViT 连接分类头 (无文字编码器)、训练参数是 Adapter+分类头, 与视频检索领域的模型在基本结构上还是有一定差异的; 从模型性能可见: 不调整文字编码器、只用 meaP 做特征融合情况下 Adapter 对性能提升作用相对有限, 只提升了 6.7%, 距离 full fine-tune 的 CLIP4Clip 的 10.9% 还有 4.3% 的差距。因此后续改进可尝试加入文字编码器、特征融合模块的改进。

04 月 12 日 (第 7 周)

内容提要:

1. 学习或进行的设计情况总结:

当前实验条件: 内存 24GB 的 GeForce RTX 3090 显卡, 1 张。

目前进行了以下 2 种实验:

- 1) AIM: 采用 CLIP-B/32、取平均值作为视频特征融合方式, 加入 Temporal Adaptation, 其余参数全部冻结;
- 2) CLIP4Clip: 采用 CLIP-B/32、取平均值作为视频特征融合方式, 无冻结参数。

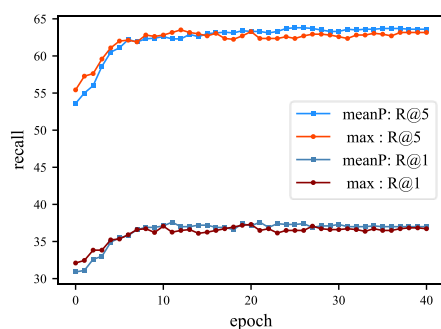
训练概况:

模型	训练集	epoch	可调参数 (M)	耗时 (hour)
AIM	MSRVTT-7K 7010×20 文本视频对	2	10.66	13.2
CLIP4Clip full fine-tune	MSRVTT-9K 9000×20 文本视频对	5	151.28	39.4

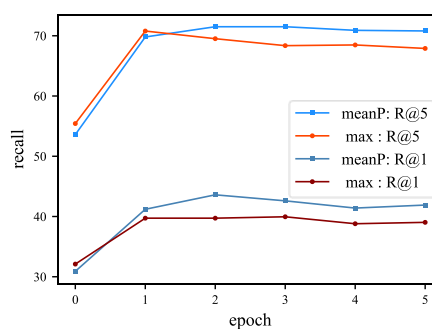
模型性能:

模型	R@1	R@5	R@10	Median R	Mean R
原始 CLIP	30.9	53.6	63.3	4.0	41.6
AIM	37.6	63.3	73.0	3.0	21.8
CLIP4Clip full fine-tune	42.2	70.2	80.9	2	15.3

2. 实验中观测到 5 分钟运行 64 step, 3 分钟时间在加载数据, GPU 利用率在 20%以下, 可以尝试将帧预先保存为图片提升数据加载速度。
3. 单独考量求和平均 (meaP) 的特征融合方式、只用相似度最高的 1 帧图片 (max) 的方式。CLIP4Clip 的 meaP 只比 max 高不到 4%, 可见 meaP 这种特征融合方式的效果并不出众; 但 AIM 的 meaP 只比 max 高不到 0.5%, 可见其时间 Adapter 确实起到了一定时间特征学习的作用, 但为何在视频检索领域里的性能提升幅度不及动作识别领域还有待进一步探究。



AIM 召回率-epoch 变化



CLIP4Clip 召回率-epoch 变化

04 月 17 日（第 8 周）

内容提要：

对于文字特征、视频特征融合模块，阅读和学习 CLIP2Video、DRL 等改进融合模块、改进损失函数的方案，尝试实现其代码并加入到当前模型中；实验 QB-Norm 后处理方案的性能；

对于视频帧采集，实现抽取视频关键帧模块，抽取关键帧并以图片形式保存，一方面可实验是否较平均抽帧有性能提升，另一方面可一定程度缓解训练时视频加载是性能瓶颈的问题（例如使用 Katna 库，其主要以 LUV 空间的颜色差异、亮度筛选、像素交叉熵、K 聚类等指标和方式筛选出不模糊的最具代表性的帧）；

对于文字、视频编码器内部，可以尝试在文字编码器中插入 LoRA，在视频编码器中补全 AIM 的 Temporal Adaptation 为 Joint Adaptation，在 full fine-tune 好的模型参数基础上，进一步进行 parameter efficient fine-tune 以提升性能；

并最终完成在 MSVD、ActivityNet、DiDeMo 等数据集 test 集上的性能验证。

04 月 24 日（第 9 周）

内容提要：

1. 平均选取方案：

这是 CLIP4Clip 论文以及其他大多数视频文本检索论文中使用的帧选取方式，按时间平均抽取：

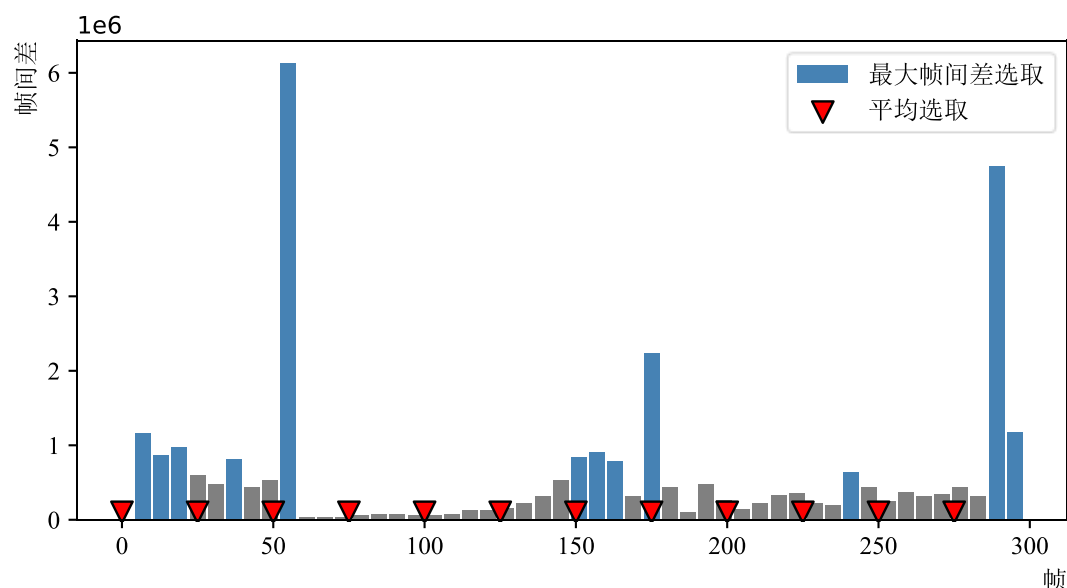
- a) 每个视频首先按秒划分，取每秒的第 1 帧；
- b) 短于 12 秒的视频关键帧数量不足 12（在 DataLoader 中用全 0 矩阵补齐至数量为 12）；
- c) 长于 12 秒的视频关键帧数量大于 12（在 DataLoader 中从中随机选取 12 个）。

2. 最大帧间差方案：

因为考虑到也许按时间平均抽取可能会无法取到能体现画面重大变化的帧，所以想探索一下其他关键帧提取方案：将所有帧按照帧间差从大到小排序，因为帧间差大表示该帧与上一帧有较大差距，那么这也就意味着视频画面出现了重大变化，因此主观上认为，这些帧能更好地集中体现视频内容：

- a) 先将视频中心裁剪为正方形以更好地贴近模型所看到的视野，然后遍历视频内所有帧；
- b) 首先将画面进行一次高斯模糊，以降低噪点的影响；
- c) 然后计算与前一帧的帧间差，将帧间差求和，并记录当前是第几帧（用于排序后将目标帧恢复回原来的时间顺序）；

- d) 然后将所有帧按照帧间差之和从大到小排序，取前 12 个即为目标帧；
- e) 将目标帧再按照其是第几帧从前往后排列，依次到原视频里取出，即得到关键帧。
3. 具体实现时，还可以使用“跳过 k 帧取一次参与计算”的方法加快最大帧间差方案的速度。以 MSR-VTT 数据集中的 video0.mp4 文件为例，两种方案提取的关键帧结果示例如下：



4. 在实验中对硬件的使用率进行检测发现，若 DataLoader 直接以视频为起点、选取关键帧、将关键帧使用 CLIP 的图片预处理流程转为模型输入，那么绝大部分时间都用于进行数据加载了，GPU 长时间处于空闲状态，这表明视频数据加载成了整个模型训练的性能瓶颈。

又因为对于同一个视频，同一种关键帧选取方案得到的关键帧总是相同的，因此考虑事先完成关键帧选取，缩放和中心裁剪成为边长 224，并将结果作为图片文件（JPG）保存起来。

这样 DataLoader 就只需要加载图片并直接转为 tensor 即可，将 OpenCV 库解析视频、关键帧选取、三线性插值的图片缩放、图片裁剪等所有 DataLoader 中需要消耗大量 IO 和 CPU 资源的过程全部去除了，可以极大地缩短 DataLoader 所耗费的时间，而且减少了很多重复计算。

05 月 08 日（第 11 周）

内容提要：

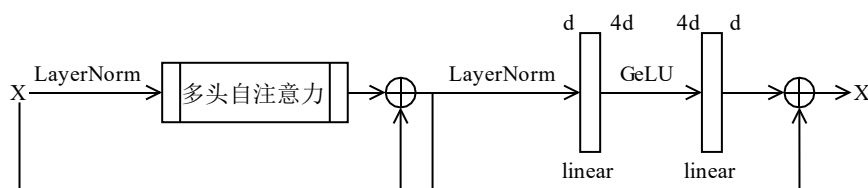
1. 原始 ViT 的 Residual Attention Block 由 1 个 MSA 层、1 个 FFN 层组成，网络输入 x 形状是 $[B \times T, N+1, D]$ ，其中 B 是 batch size， T 是一段视频取的关键帧数量数（ $T=12$ ）， N 是一张图片的图像块数量（ $N=49$ ）， D 是特征宽度（ $D=768$ ）。Adapter

插入方式如下：

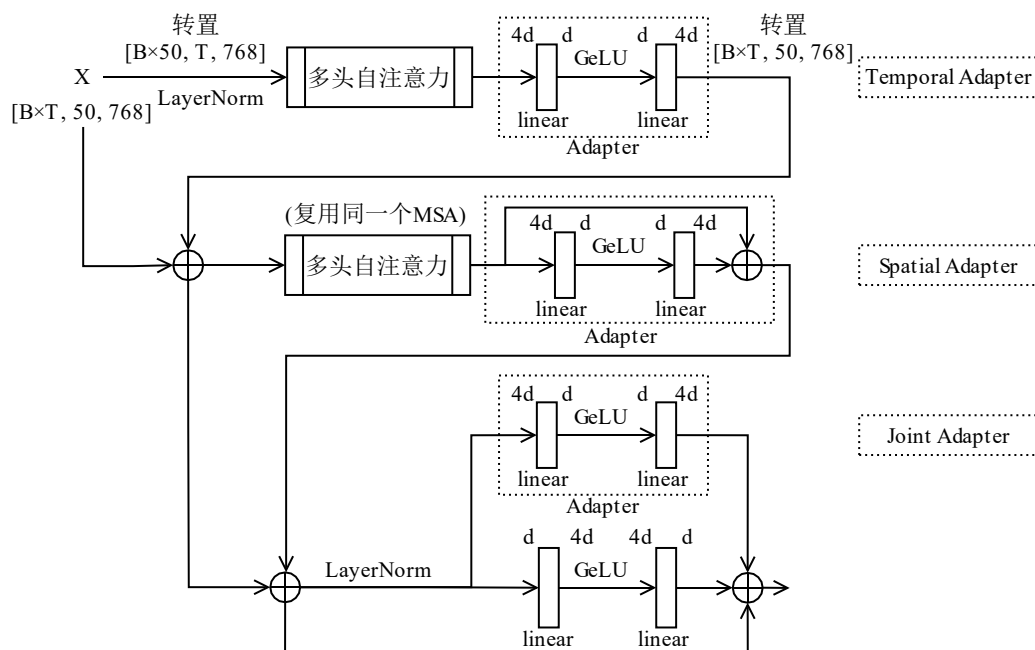
- Temporal Adapter**，插入到 MSA 层之后。先将 x 转置为 $[B \times (N+1), T, D]$ 并进行 layer norm，然后依次输入 MSA 和 Temporal Adapter，然后与 x 相加得到 x_t （因为 ViT 的 MSA 旁有跳跃连接）；
- Spatial Adapter**，插入到复用的 MSA 层之后。先将 x_t 转置回 $[B \times T, N+1, D]$ ，然后依次输入 MSA 和 Spatial Adapter，然后与 x_t 相加得到 x_s （因为 ViT 的 MSA 旁有跳跃连接）；
- Joint Adapter**，串联到 FFN 旁边。先将 x_s 进行 layer norm，然后分别输入到 Joint Adapter 和 FFN，将 2 个输出之和 x_s 相加得到最后的输出 z （因为 ViT 的 FFN 旁有跳跃连接）。

- 为了使修改后的模型初始化时模型输出与原始模型一致，所以所有 Adapter 内 FC2 权重初始化为 0；Temporal Adapter 和 Joint Adapter 不使用跳跃连接；Spatial Adapter 使用跳跃连接。

- 原始 Transformer 层结构如下：



使用 AIM 的 Adapter 插入方案，插入 Adapter 后的结构如下：



- 进行的实验：

- 训练速度**：原始 CLIP4Clip、CLIP4Clip+帧保存、AIM+帧保存，三种方案的速

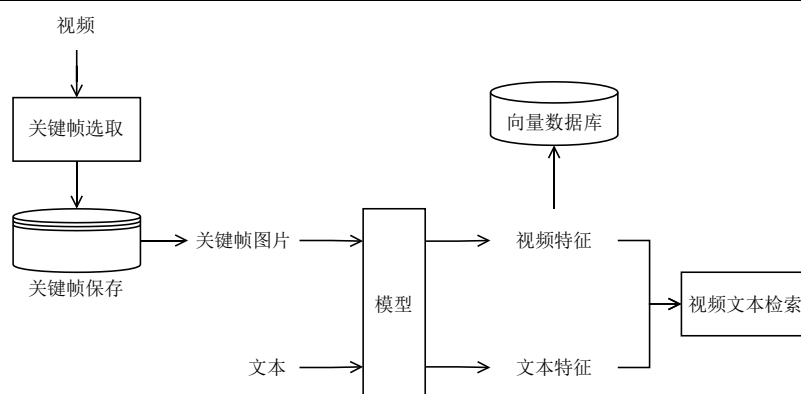
度对比；

- b) 关键帧选取方案：分别使用平均选取方案和最大帧间差选取方案抽取训练集关键帧，训练得到的模型，分别在使用平均选取方案和最大帧间差选取方案抽取关键帧的测试集上测试得分；
- c) AIM 有效性：需要证明动作识别领域的 AIM，其 Adapter 插入方案在视频文本检索领域是否有效，使用 CLIP 和 CLIP4Clip 作为对照，测试以 CLIP 为预训练模型的情况下，只训练 AIM 和文本编码器，检验是否和 CLIP4Clip 具有相近的性能；
- d) 测试 AIM 是否可用于提升模型性能，以 CLIP4Clip 为预训练模型，再次微调，检验是否有性能提升；
- e) 在 MSVD 数据集上分别以 CLIP、CLIP4Clip 为预训练模型，测试 AIM 的有效性。

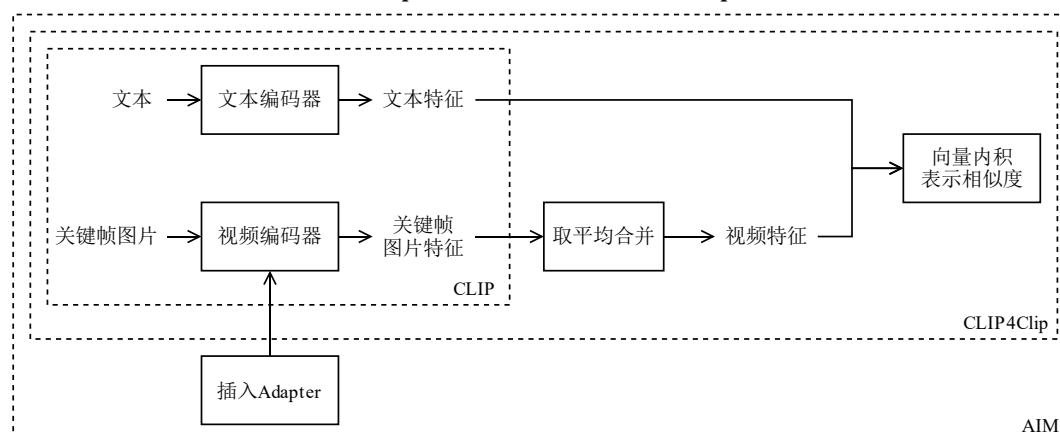
05 月 15 日（第 12 周）

内容提要：

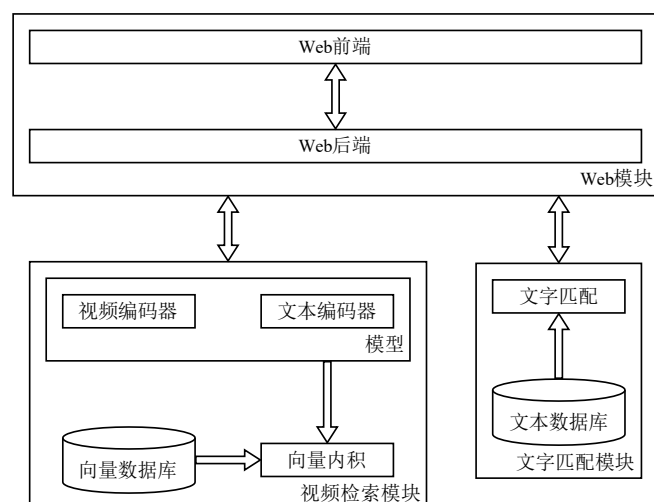
1. 论文结构：论文共分为五章。
 - a) 第一章绪论，主要介绍了目前视频文本检索领域和低参数量微调的主流方法及最新模型，并结合本实验的具体情况，分析了各个低参数量微调方案在本实验条件下的优点和缺点，提出了本文方法的依据和解决思路。
 - b) 第二章为相关技术介绍，结合各自的论文，介绍了实验中使用的 CLIP、CLIP4Clip、AIM 三个模型。
 - c) 第三章为算法设计与实现。首先提出算法，介绍了算法的整体结构，详细介绍了模型的结构与实现，并从原因、实现方法和优势等方面介绍两种关键帧采样方法、关键帧保存方案、向量数据库。然后进行多组实验对算法各个部分进行测试，详细介绍了加快模型训练、关键帧选取方案、AIM 有效性、提升模型性能、不同数据集得分等五个方面的实验及结果，并给出了相应结论。
 - d) 第四章系统设计与实现，按照需求分析、总体设计、详细设计、实现、测试的逻辑组织顺序介绍了所设计的视频文本检索系统，给出了系统需求分析、架构、流程图、用例图，介绍了系统各个模型的功能、意义及实现。
 - e) 第五章结论，总结了实验内容，给出了不足和展望。
2. 算法整体结构。算法主要由关键帧选取、关键帧保存、模型、向量数据库四个部分组成，其中：



- 关键帧选取获得每个视频的图像，共有平均选取和最大帧间差选取这两种方案；
 - 关键帧保存将视频的关键帧以图片形式保存，以加快训练速度；
 - 模型是使用的网络模型，可以对文本编码获得文本特征，对视频的关键帧图片编码获得视频特征；
 - 向量数据库将视频特征保存起来，这样模型在测试时，便可以直接从向量数据库加载视频特征，而无需重新计算，从而加快模型在测试时的速度。
3. 模型整体结构：实验中，使用的是 CLIP 系列模型最小的 CLIP ViT-B/32 作为模型主体，采用 CLIP4Clip 中将视频转化为 CLIP 输入的思路构造实验所用模型，并且实验所用模型可采用 AIM 的 Adapter 插入策略进行 Adapter 插入。



4. 系统框架。文本视频检索系统主要包括以下三个模块：



- a) 视频检索模块：主要包括模型和向量数据库两部分，针对满足“检索速度”的功能性需求而设计。模块接收文本，使用模型的文本编码器将文本编码为文本特征；然后与向量数据库中保存的（视频库中所有视频的）视频特征计算相似度（使用矩阵内积作为相似度指标）；并返回相似度最大的 k 个视频，作为视频检索结果。
- b) Web 模块：使用 Python 中的 Django 库搭建，主要包括 Web 前端和 Web 后端两部分。其中 Web 前端接收用户输入传递给 Web 后端，并接收 Web 后端传递的 HTML 界面信息进行展示；Web 后端处理 Web 前端传来的用户输入，然后调用相应视频检索模块和文字匹配模块计算得到结果，并将结果处理为 HTML 界面，返回给 Web 前端。
- c) 文字匹配模块：为满足让用户“可选择展示文字匹配结果作为参考”的功能性需求而设立的模块，主要包括文本数据库。文本数据库使用 SQLite 数据库（传统的关系型数据库）存储了视频库中所有视频的视频描述（即 MSR-VTT 数据集中的文字描述）。当用户开启文字匹配模式时，模块接收文本，并使用文本在文本数据库进行文字匹配；返回视频描述含有该文本的视频，作为文字匹配结果。

05 月 22 日（第 13 周）

指导教师签字：



2023 年 6 月 15 日