廣東財經大學
GUANGDONG UNIVERSITY OF FINANCE & ECONOMICS

广东财经大学		2024-JX16-

本科毕业论文（设计）

基于 Bert 的专利 IPC 分类设计与实现

学　　院	信息学院
专　　业	计算机科学与技术
学　　号	20251102150
学生姓名	黄瑞杰
指导教师	沈永珞
提交日期	2024 年 4 月 5 日

毕业论文(设计)成绩评定表

毕业论文（设计）指导教师评语及成绩 成绩_____　　指导教师签名_____ 年　　月　　日
毕业论文（设计）复评教师评语及成绩 成绩_____　　复评教师签名_____ 年　　月　　日
毕业论文（设计）答辩评语及成绩 成绩_____　　答辩委员会主席签名_____ 年　　月
毕业论文（设计）总成绩（五级记分制）_____　　院（系）负责人签名_____ 年　　月　　日

毕业论文题目：《基于 Bert 的专利 IPC 分类设计与实现》

计算机科学与技术 专业 2020 级 本科生姓名：黄瑞杰

指导教师（姓名、职称）：沈永珞副教授

内容摘要

随着时代进步，我国国内专利商标拥有量稳步上升，拥有专利的创新型企业数量增长较快，专利申请量一直稳居世界第一。IPC 分类是对专利文档进行分类的重要标准，但传统分类方法如人工分类和机器学习分类存在一定局限性。因此，本设计提出基于 Bert 的自动分类方案，旨在利用 Bert 模型实现专利 IPC（国际专利分类）的自动分类。通过数据预处理、模型构建、训练和评估等步骤，实现了该方案。实验针对专利 IPC 的部和大类进行训练，结果实现 A-H 部 0.96572、H 部大类 0.96588 和 A-H 部大类 0.69252 的精确度，基本能够实现精确分类的需求。总结分析指出了该方法的优点和改进空间，为专利检索和管理提供了新的思路和方法。

关键词：Bert 模型；专利 IPC 分类；文本特征提取；深度学习

TITLE: Design and implementation of patent IPC classification based on Bert

MAJOR: Computer Science and Technology

APPLICANT: RuiJie Huang

SUPERVISOR: Yongluo Shen

Abstract

With the progress of the times, the number of domestic patents and trademarks in China has increased steadily, the number of innovative enterprises with patents has grown rapidly, and the number of patent applications has ranked first in the world. IPC classification is an important standard for classifying patent documents, but traditional classification methods such as manual classification and machine learning classification have certain limitations. Therefore, this design proposes an automatic classification scheme based on Bert, which aims to realize the automatic classification of patent IPC (International Patent Classification) by using the Bert model. The scheme is implemented through data preprocessing, model construction, training, and evaluation. The results of the experiment were carried out on the parts and classes of the patent IPC, and the results achieved the accuracy of 0.96572 in A-H, 0.96588 in 0.96588 and 0.69252 in A-H, which can basically meet the requirements of accurate classification. The summary and analysis points out the advantages and improvement space of the method, and provides new ideas and methods for patent search and management.

Key words: Bert model; International Patent Classification (IPC) classification; Text feature extraction; Deep Learning

目　　录

1. 引言

1.1 研究背景

专利（Patent）的起源可以追溯到中世纪的英国，在那个时候，个人引进外国技术后会获得一种专利证，以获得对该技术的独占权。专利是指一种专有的权利和利益，由法律规范保护，并得到国家的认可。当发明人根据专利法规定向专利主管机关提出申请时，如果符合法律规定的条件，他们将在规定的时间内获得对其申请的专利产品的占有权。在全球范围内，中国是专利申请量最多的国家，位居第一，美国位居第二，拥有58.9万件专利；日本排名第三，拥有31.8万件专利[1]。在当今高度信息化的社会，专利技术在创新和经济发展中起着举足轻重的作用。专利分类作为专利信息管理和检索的重要手段，对于促进技术交流、引导创新方向、保护知识产权等方面具有重要意义。然而，传统的专利分类方法受限于人工标注的成本和主观因素，难以满足快速增长的专利数据处理需求，借助Bert模型有助于提高专利分类的效率和准确性，推动专利信息管理和知识产权保护工作的进步。因此，借助自然语言处理、深度学习等技术进行自动化处理和改进对专利文本的分类具有重要意义。

1.2 研究现状

近年来，深度学习技术的发展对专利分类任务带来了新的突破。其中，使用预训练的语言模型（如 BERT、GPT）进行特征提取和分类的方法备受关注，也逐渐被应用于专利分类任务中。就目前深度学习专利分类的情况而言，常见的模型包括卷积神经网络（CNN）、循环神经网络（RNN）、长短期记忆网络（LSTM）和Transformer等。这些模型具有学习文本语义表示的能力，可应用于各种文本分类和分析任务，并在许多自然语言处理任务中取得了很好的效果。Transformer是一种由Vaswani等人在2017年提出的一种新型神经网络结构，对处理序列数据具有革命性的影响，特别在自然语言处理（NLP）领域表现突出。BERT模型建立在Transformer的基础上，通过双向编码器（Bidirectional Encoder）来学习文本中的语言表示。这种结构使得BERT在各种自然语言处理（NLP）任务中展现出了十分优秀的性能。而基于Bert的专利分类研究是相较于之前机器学习和深度学习分类方式的优化，通过实现本项目的研究，有助于推进新型自然语言技术的发展，对实现自动化分类提供帮助。

1.3 研究方法

项目是基于中文的分类任务，使用到的是bert-base-chinese模型，这是一种基于Transformer 架构的预训练语言模型，它是 Google 在 2018 年发布的 BERT

（Bidirectional Encoder Representations from Transformers）模型的一个变种，被广泛应用于各种中文自然语言处理任务中，本项目就是通过其模型对中文专利进行 IPC 分类研究。

首先，进行数据收集，数据集为 CSV 格式。收集本研究的专利文本数据集为 A-H 部整合的数据，独立的 H 部 H01-H05 的数据，用于完成三种分类，其一是 A-H 部的 IPC 部分类，其二为针对 H 部的专利 IPC 前三位的分类，最后即对 A-H 部的专利 IPC 前三位的分类（除去 A-H 部中其他类目中不包括的技术主题部分，例 A99、B99 等等）。

接着，即使用 Bert 模型对数据的处理和应用模型的使用。对数据处理时，首先读取数据，用 panda 对文件数据进行提取；接着用了一个基于 Dataset 的类对收集的数据进行处理，即将文本处理成 bert 的输入格式的数据，并将 bert 的输入格式的数据利用 Dataset 封装；模型是 BERT-based Models 应用模型，对其进行封装微调，用到 SequenceClassifierOutput 返回其模型输出.

最后，在进行模型的训练和评估时，首先需要载入数据。随后，我们会初始化一个自定义的模型，这个模型是基于已有的神经网络结构进行定制化设计的。接着，我们会将模型加载到 GPU 上，以便在其上进行训练，这样可以加快训练速度并提高效率。

1.4 研究目的和意义

随着自然语言处理技术的发展，特别是基于预训练模型的方法（如 BERT），在文本分类任务中取得了显著的性能提升，为专利文本分类提供了新的思路和技术手段。而传统的专利分类方法通常依赖于人工标注和专家经验，或者是使用旧的分类模型进行分类，效率普遍不高，如果能用当前最新分类模型进行分类，就能满足快速增长的专利数据需求，来提高专利分类效率。

通过使用 Bert 预训练模型对专利 IPC 进行分类，不仅有助于提高专利分类的效率和准确性，推动专利信息管理和知识产权保护工作的进步，也有助于促进技术交流与创新合作，推动自然语言处理技术在实际应用中的落地，从而推动经济社会的发展。

2. 相关研究综述

2.1 专利 IPC 分类研究

专利 IPC 指的是国际专利分类（International Patent Classification），是一种用于对专利文献进行分类和组织的国际标准系统。IPC 由世界知识产权组织（WIPO）管理，它将技术领域划分为不同的部门、类别和子类别，以便对专利文献进行系统化的组织和检索。

IPC 的结构基于技术的主题，采用了一种层次结构，包括部、类、子类和组。IPC 将技术领域划分为不同的部门，如 A 部表示人类生活必需，B 部表示作业、运输；然后每个部门下面进一步划分为类别和子类别，具体描述了技术领域的细分主题。例如，A 部可以包含大类别 A01，表示的是农业。因此，通过 IPC 可对专利文献进行分类、检索和组织。当申请专利时，申请人需要为其专利文献指定一个或多个适当的 IPC 分类码，以确保该专利与相关技术领域的其他专利文献得到正确的分类和检索。对于专利检索人员和研究人员来说，IPC 分类码也是进行专利检索和分析的重要工具之一，如果能更加迅捷的方式识别专利 IPC，就既能对专利有效分类，还可方便人们进行检索。

2.2 Python 网络爬虫

2.2.1 网络爬虫

在信息化时代，互联网已经成为人们获取和共享信息的主要途径。海量数据广泛分布在各种网站和平台上，包括社交媒体、电商平台、学术资源和新闻网站。如此庞大且复杂的数据量远远超出了传统手工收集和处理的能力，因此，自动化的数据抓取和处理技术变得尤为重要。

网络爬虫（Web Crawlers），也被称为网络蜘蛛（Web Spiders）或网络机器人（Web Robots），是一种自动化工具，能够系统性地浏览和收集网络数据。通过访问公开网页，网络爬虫可以提取有价值的信息并存储到本地数据库，为后续的数据分析和处理奠定基础。由于 Python 的简洁易用及其丰富的库支持，它成为构建网络爬虫的理想语言。

2.2.2 爬虫设计方式

基于 Python 的爬虫设计利用 Python 编程语言及其丰富的第三方库，构建自动化的工具来浏览和收集网络数据。Python 因其简洁的语法和强大的库支持，被广泛应用于网络爬虫开发。主要的库包括：用于发送 HTTP 请求的 requests 库，用于解析 HTML 内容的 BeautifulSoup、lxml 和 html.parser 库，以及用于处理和存储数据的 pandas 库。此外，Scrapy 框架提供了构建复杂和可扩展爬虫的高级功能。

设计一个基于 Python 的爬虫，首先需要发送 HTTP 请求获取网页内容，接着处理分页和动态内容，之后通过反爬虫策略和遵守网络规则避免封禁，再则进行一定的错误处理和日志记录来调试和监控，最后根据需求选择性优化。Python 的爬虫设计涵盖了 HTTP 请求、HTML 解析、数据处理和存储等多个方面。通过合理选择库和工具，遵循网络礼节和反爬虫策略，处理分页和动态内容，并进行有效的错误处理和日志记录，可以开发出一个功能强大且高效的爬虫系统。这个系统不仅能够高效地收集和处理大量网络数据，还能为后续的数据分析和研究提供坚实的基础。

2.3 Bert 模型

Bert 模型基于 transformer，它是一个预训练模型，且与传统的单向语言模型不同，Bert 模型使用了双向编码器，它可以同时考虑一个词的左右上下文信息，从而更好地捕捉词语在句子中的含义。Bert 模型的预训练过程包括两个任务：掩盖语言模型（Masked Language Model，MLM）和下一句预测（Next Sentence Prediction，NSP）。在 MLM 任务中，模型需要预测句子中被随机掩盖的一些词语；在 NSP 任务中，模型需要判断两个句子是否是连续的上下文关系。通过这两个任务的预训练，BERT 模型可以学习到丰富的文本语义表示，从而在各种自然语言处理任务中取得出色的性能。

Bert 模型只使用了 Transformer 的编码器（encoder）部分，其整体框架是由多层 Transformer 的编码器堆叠而成的。每一层的编码器由一层多头注意力（multi-head attention）和一层前馈神经网络（feed-forward network）组成。大型的 BERT 模型通常包含 24 层编码器，每层有 16 个注意力头；而小型的 Bert 模型包含 12 层编码器，每层有 12 个注意力头。在 BERT 模型中，每个注意力（attention）头的主要作用是通过目标词与句子中的所有词汇的相关度，对目标词进行重新编码。因此，每个注意力头的计算过程通常包括三个步骤：计算词之间的相关度（Attention Score），对相关度归一化（Attention Weights Normalization），加权求和获取目标词的编码（Weighted Sum for Encoding）。在通过 attention 计算词之间的相关度时，首先通过三个权重矩阵对输入的序列向量(512*768)做线性变换，分别生成 query、key 和 value 三个新的序列向量，用每个词的 query 向量分别和序列中的所有词的 key 向量做乘积，得到词与词之间的相关度，然后这个相关度再通过 softmax 进行归一化，归一化后的权重与 value 加权求和，得到每个词新的编码。以上这一过程可以有效地捕捉目标词与句子中其他词之间的关系，从而为模型提供更丰富的语义信息。

2.3.1 模型输入

在 Bert 中，输入向量由三种不同的 embedding 求和而成，如下图所示，由单词本身向量（Token Embeddings）、区分两个句子的向量（Segment Embeddings）、单词信息位置编码的特征向量（Position Embeddings），三者相加即 Bert 模型输入。

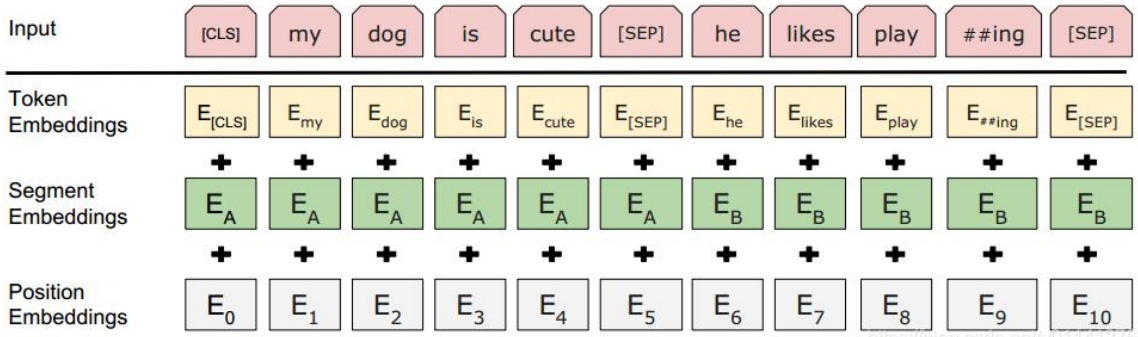

图 2-1 输入结构图

Figure 2-1 Input structure diagram

根据这样的 BERT 模型的输入，即 X=(batch_size, max_len, embedding)，假设 batch_size=1，输入的句子长度为 128，每个词的向量表示的长度为 512，那么整个模型的输入就是一个 128*512 的 tensor。

2.3.2 网络结构

Bert 模型基于 transformer，但不完全包含其结构。一个 Bert 预训练模型的基础结构是 transformer 结构的 encoder 部分，由输入部分+多个 encoder 的网络+输出部分组成，如下图 2-2，是 Bert 模型的网络结构图。

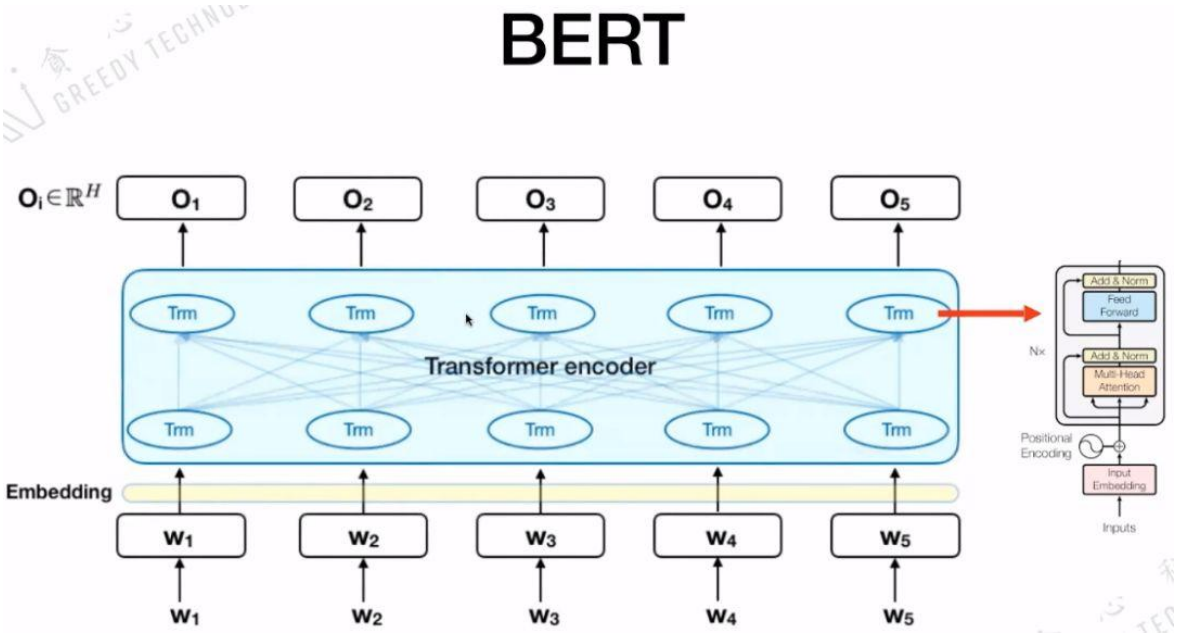

图 2-2 网络结构图

Figure 2-2 Network Structure Diagram

对一个 transformer 的 encoder 单元细化分，是 multi-head-Attention + Layer Normalization + Feedforword + Layer Normalization 的叠加。当然，BERT 的每一层都由一个这样的 encoder 单元构成[2]，而一个 encoder 模式就如下图 2-3。

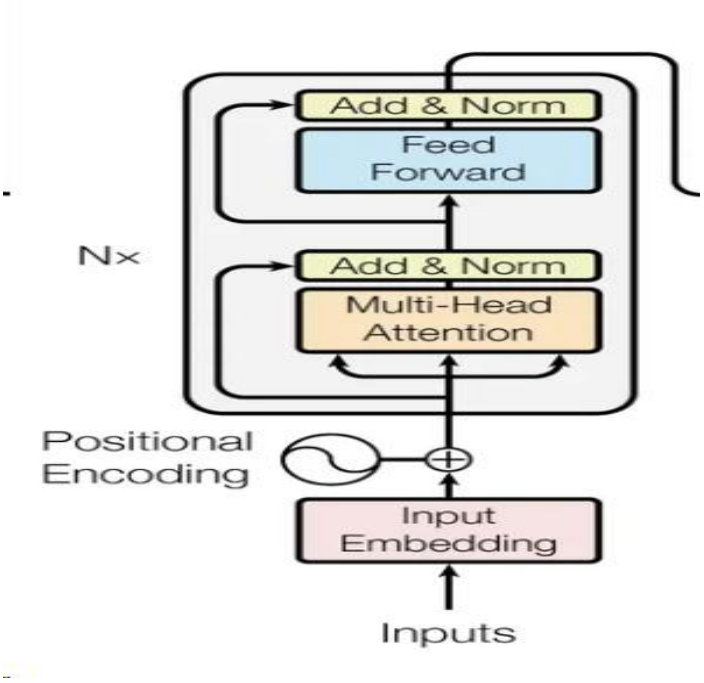

图 2-3 编码器模块

Figure 2-3 Encoder Module

2.3.3 Self-Attention Layer

self-attention 也称自注意力机制，是在 Transformer 模型中的一个关键组件，也是多头注意力的组成单元，它使模型在处理每个标记时能够专注于输入序列的不同部分。单个自注意力头的计算过程如下。

首先将输入数据转换成矩阵，对于给定的输入序列 X，首先通过三个线性变换（通常是全连接层）将其转换为查询（Q）、键（K）和值（V）向量[3]。这些转换的目的是将原始输入映射到不同的表示空间，以便后续计算。上面三个线性变换相当于输入序列 X 与三个不同的矩阵相乘得到 Q、K、V。

其次计算注意力分数，对于每个查询向量 Q，计算它与所有键向量 K 的转置矩阵之间的点积。这将得到一组注意力分数[4]，表示了 Q 与输入序列中每个位置的相关性。接着缩放注意力分数，即除以这组注意力分数维度的均方根，为了控制梯度消失的问题，通常对注意力分数进行缩放，以确保它们的大小在一个合理的范围内，防止结果过大。

然后直接应用 Softmax 函数，将缩放后的注意力分数输入到 softmax 函数中，以获得每个位置的注意力权重。Softmax 函数确保所有权重的总和为 1，表示每个位置对查询的相对重要性。

最后进行加权求和，使用注意力权重对值向量 V 进行加权求和，以获得最终的自注意力输出。每个位置的值向量被赋予了由注意力权重决定的重要性，从而产生了针对查询的加权表示。

用数学表达式来表示如下：

$$Attention(\mathrm{Q},\mathrm{K},\mathrm{V}) = softmax(\frac{\mathrm{Q}*\mathrm{K}^{\mathrm{T}}}{\sqrt{\mathrm{d_k}}}) * \mathrm{V} \quad (1)$$

其中，Q，K，V 分别是查询、键、值向量，d_k 是查询向量的维度。softmax 函数应用在每行上，以获得注意力权重。

2.3.4 multi-head-Attention

多头注意力（Multi-head Attention）是一种用于处理序列数据的注意力机制，是在自注意力机制的基础上进一步发展的。实际上是多个并行的 Self-Attention 层，每个“头”都独立地学习不同的注意力权重，它通过并行地应用多个不同的注意力头（attention head）来增加模型的表达能力和学习能力。

在多头注意力机制中，首先将输入序列分别映射到多个子空间，每个子空间都对应着一个注意力头。然后，每个注意力头都计算出一组注意力权重，这些权重用来加权求和输入序列的不同部分，从而产生多个加权的表示。最后，通过将这些加权表示串联或者拼接起来，得到最终的多头注意力输出[5]，如下图所示。

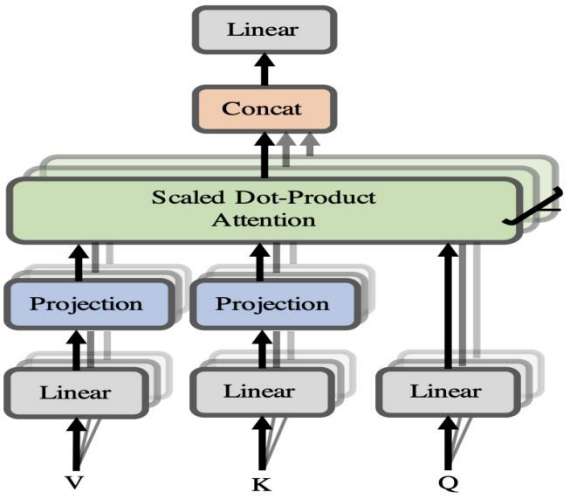

图 2-4 多头注意力图

Figure 2-4 Multi head attention map

2.3.5 Layer Normalization

Transformer 里每一个子层（self-attetion，Feed Forward Neural Network）之后都会接一个残缺模块，且有一个 Layer normalization，具体位置即图 2-3 的 Add & Norn 部分。Layer Normalization 是一种用于神经网络中的归一化技术，它类似于 Batch Normalization（批归一化），但是在归一化的对象和计算方式上有所不同。

在 Layer Normalization 中，归一化是在每个样本的特征维度上进行的，而不是在批次维度上进行。具体来说，对于给定层的每个单元（神经元），Layer Normalization 计算其输入特征的均值和标准差，并用这些统计量对该单元的输入进行归一化。这样，每个单元都独立地归一化了，而不是像批归一化那样对整个批次进行归一化。

Layer Normalization 的公式如下：

$$LN(x) = \frac{x-\mu}{\sigma} * \gamma + \beta \quad (2)$$

其中，x 是输入向量；μ 是输入向量的均值；σ 是输入向量的标准差；γ 和 β 是可学习的缩放和偏移参数，用于恢复数据的表示能力。

2.3.6 Feed-forword

Feed-forward 通常指的是 Transformer 模型中的前馈神经网络（Feed-forward Neural Network），包含在每个 transformer 的编码器层里。前馈神经网络一般由两个线性层和一个激活函数组成，这些层通常是全连接层，即一个前馈神经网络 = 第一个全连接层 + 非线性激活函数 + 第二个全连接层[6]。

首先是第一个全连接层接收输入向量，并将其映射到一个更高维度的中间表示空间，接着非线性激活函数常见的选择是 ReLU（Rectified Linear Unit）激活函数。ReLU 激活函数能够引入非线性，帮助网络学习到更复杂的特征。最后第二个全连接层接收第一个全连接层的输出，并将其映射回原始输入的维度空间，或者其他合适的维度。由于中间具有一个非线性激活函数，所以两个全连接层在一般情况下是不包含激活函数的，同时两个全连接层都是线性变换的，以上即前馈神经网络的要点。

在 BERT 模型中，在预训练的 BERT 模型中，并没有显式地使用“Feed-forward”层，因为 BERT 模型是基于 Transformer 架构构建的，其中的前馈神经网络（Feed-Forward Neural Network）在每个 Transformer Encoder 层中被使用。

2.3.7 Bert 模型优缺点

优点上，Bert 模型基于 Transformer 架构，可以进行双向的语言理解，也就是相对 RNN[7]更加高效、能捕捉更长距离的依赖。该模型采用两个任务进行预训练，分别是遮盖语言建模（Masked Language Modeling, MLM）和下一句预测（Next Sentence Prediction, NSP），在文本分类任务中，只需要在 Transformer 的输出之上加一个分类层。对比起之前的预训练模型，会更加高效。

Bert 模型的一个缺点是与预训练阶段的 MASK 问题即文本标记相关。在预训练时，只有每个批次中约 15%的文本被预测，而且在实际预测中，MASK 标记并不会出现。因此，训练时使用过多的 MASK 可能会影响模型的性能。此外，在训练时，需要确保训练数据被打乱，以及检查每个批次是否包含多个类别的样本。如果一个批次中只包含一个类别的样本，则模型将无法有效地进行优化，由此可得，Bert 模型对数据的收敛和预测具有一定的要求。但在数据量庞大的情况下，也可以通过其他样本来弥补被 MASK 标记掩盖的文本之间的关系。尽管存在一些限制，但在大规模数据集上，Bert 模型仍然可以通过学习到的样本之间的关系来提高性能。

3. 数据与预处理

3.1 数据源与爬虫

数据来源于专利汇（PATENTHUB），通过数据收集，得到的每一条数据包含专利号、专利名称、摘要、IPC 分类号、法律状态等五项内容。一共两份数据，专门供 H 部分类的数据集 50001 条，供部分类和全部部的大类分类的数据集共 62199 条网页如下图所示。

图 3-1 专利汇网页

Figure 3-1 PATENTHUB Website

在搜索框里输入“H01”进行搜索，打开网址后为下一步处理做准备，如图 3-2

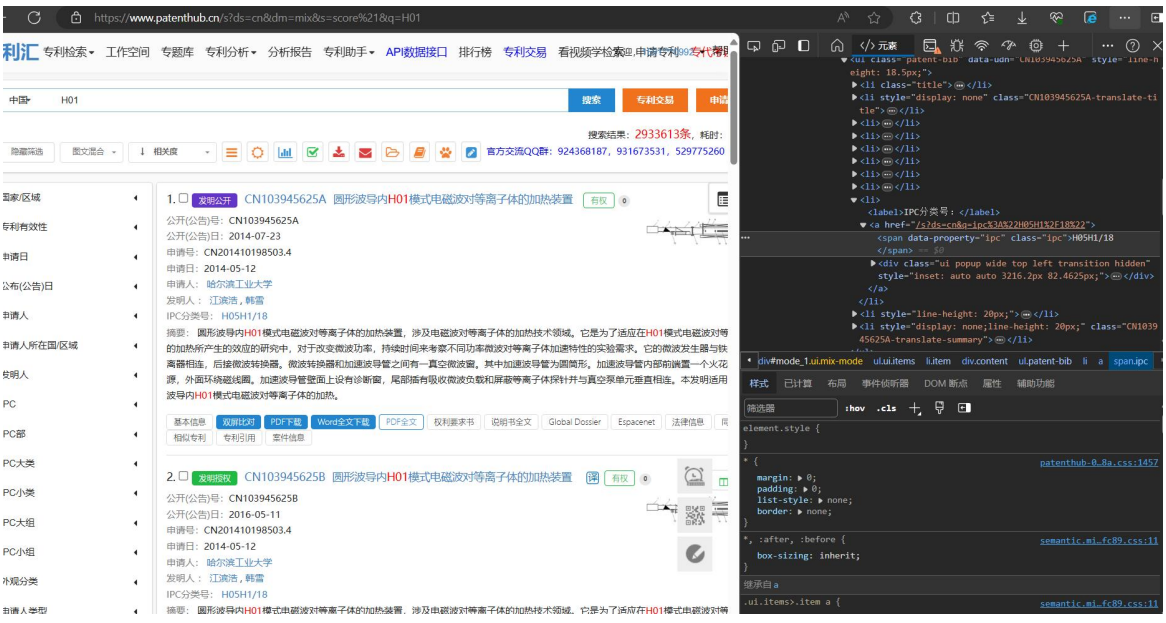

图 3-2 专利汇网页结构

Figure 3-2 PATENTHUB Website Structure

项目利用了 Python 编程语言以及一系列强大的第三方库，如 requests、pandas 和 BeautifulSoup。通过发送 HTTP 请求获取网页内容，然后使用 BeautifulSoup 库解析

HTML 内容并提取所需的数据，最后使用 panda 库将提取的数据保存为 csv 格式。

3.2 数据集

收集了两种不同格式的专利文本数据集，一个是 A-H 部整合的数据集，另一个是独立的 H 部（H01-H05）的数据集。数据集分为三种分类任务：A-H 部的 IPC 部分类、针对 H 部的专利 IPC 前三位的分类、A-H 部的专利 IPC 前三位的分类（排除不包含在其他类目中的技术主题部分，如 A99、B99 等）。其中，由于 A 部有 15 个类，B 部有 37 个类，C 部有 20 个类，D 部有 8 个类，E 部有 7 个类，F 部有 17 个类，G 部有 14 个类，H 部有 6 个类，共 124 个类，每个部数目不同，因此收集的数据数目按比例有所不同，用柱状图表示 A-H 部的数据集如下图 3-3。

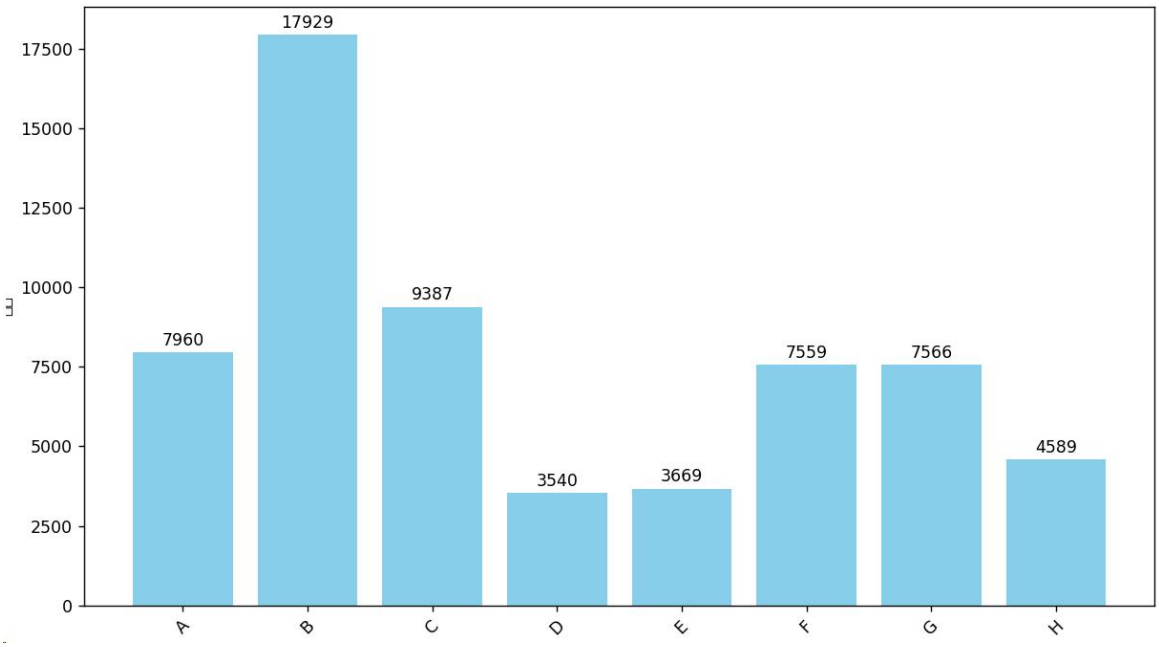

图 3-3 A-H 部数据集

Figure 3-3 A-H Dataset

同时项目单独对 H 类进行分类，以 H 部（H01-H05）为例，目的是比较 Bert 模型在不同分类数下的分类效果。收集的 H 部每个类的数据都为 10000 条，以便于训练。

以上两份收集好的数据都是按 IPC 分类号顺序排布的，以总数据集为例，通过收集得到的顺序是从 IPC 号 A01 开始顺序排布的，而且所有数据都是以专利号、专利名称、摘要、IPC 分类号、法律状态五列数据进行排布，如下图 3-4 的数据样式。

专利号	专利名称	摘要	IPC分类号	法律状态
CN11138...	一种组合...	本发明公...	A01G9/0...	无效
CN11138...	一种拼插...	本发明公...	A01G9/02	无效
CN11137...	一种梅花...	本发明公...	A01G17/14	无效
CN11137...	一种梅花...	本发明公...	A01C5/02	无效
CN10972...	一种可调...	本发明涉...	A01G9/14	有权
CN11137...	一种自动...	本发明公...	A01G27/...	无效
CN11137...	一种梅花...	本发明公...	A01G27/...	无效
CN11137...	一种使花...	本发明公...	A01G7/04	无效
CN10972...	一种轻便...	本发明涉...	A01G9/14	无效
CN10972...	一种可调...	本发明涉...	A01G9/14	有权
CN11138...	一种循环...	本发明公...	A01G9/02	无效
CN11137...	一种园林...	本发明公...	A01M7/00	无效
CN11137...	一种梅花...	本发明公...	A01G9/14	无效
CN10969...	装配式垂...	本发明提...	A01G9/02	无效
CN2093...	一种园林...	本实用新...	A01M7/00	失效
CN2093...	一种自动...	本实用新...	A01G27/...	失效
CN2093...	一种梅花...	本实用新...	A01G9/14	失效
CN2093...	一种梅花...	本实用新...	A01C5/02	失效
CN2095...	一种轻便...	本实用新...	A01G9/14	有权

图 3-4 数据样式

Figure 3-4 Data Style

3.3 训练集和测试集划分

划分 BERT 模型的数据集为训练集和测试集的方法和原则与一般的机器学习任务类似。首先在划分数据集之前，应该确保数据集的随机性。这意味着要先对数据进行随机化处理，以防止数据的排列顺序对模型的训练和测试产生影响；接着与一般的机器学习任务一样，划分数据集时应该保持训练集和测试集中的数据分布相似，确保每个类别在训练集和测试集中的分布大致相同，项目收集是按照类比例搜集的，符合要求；然后确保训练集和测试集之间没有数据泄漏的问题，数据泄漏可能会导致模型在测试集上的性能评估不准确，本研究即确保每个类都有具体数据，在收集阶段完成；最后分集，将数据集划分为训练集和测试集，训练集用于模型的训练，而测试集则用于最终评估模型的性能[8]。

另外一个要点就是保证数据量，本项目收集的数据充分，确保了训练集和测试集的数据量足够大，以确保模型可以充分学习和泛化。

对于整体数据的划分，项目使用了 pandas 库来加载和处理数据，然后使用了 sample 方法来随机打乱数据集并分离出训练集和验证集，训练集占总数据集的 80%，测试集占 20%。以下是脚本的基本思路：首先从 CSV 文件中加载数据；接着用 sample 打乱数据集；然后将整体的数据集进行划分，得到两份数据做为训练集和测试集；最后将训练集和测试集保存为新的 CSV 文件。数据集处理代码如下。

```
import pandas as pd
data = pd.read_csv('总分类/data/pre_data.csv')
# 打乱数据集
data = data.sample(frac=1).reset_index(drop=True)
#训练集和测试比例
```

```
train_ratio = 0.8
val_ratio = 1 - train_ratio
# 数量
train_size=int(train_ratio*len(data))
val_size=len(data)-train_size
#分离数据集
train=data.sample(n=train_size,random_state=42)
val=data.drop(train.index)
#保存
train.to_csv('pre_data.csv',index=False)
val.to_csv('val_data.csv',index=False)
```

4. 模型设计与实现

4.1 输入处理

Bert 的输入是三个 embedding 的求和，即 token embedding、segment embedding 和 position embedding。假设需要输入一句话是“我爱你，你爱我”，我们需要利用 tokenizer 进行初步的 embedding 处理，得到的编码如下：input_ids：[101, 2769, 4263, 872, 102, 872, 4263, 2769, 102]，其中 101 表示[CLS]开始符号，102 表示[SEP]句子结尾分割符号，其他数字对应句子中的具体词语；token_type_ids：[0, 0, 0, 0, 0, 1, 1, 1, 1]，用于区分上下句，上句编码为 0，下句编码为 1；attention_mask：[1, 1, 1, 1, 1, 1, 1, 1, 1]，指定哪些词语作为 query 参与到 attention 操作中，所有词语都被指定为 1，表示所有词都作为 query 计算与其他词的相关度。

在使用 Bert 模型过程中，本人自己设计了一个 embedding 方法，即利用 Python 库 pandas 对 CSV 格式的数据进行提取，并基于 PyTorch 中的 Dataset 抽象类对数据进行处理。通过继承 Dataset 类创建自定义的数据集类 InputDataSet，以便加载和处理数据，将文本转换为 Bert 模型的输入格式。例如，在处理一个文本数据集时，我们可以通过 pandas 读取 CSV 文件中的数据，并利用 tokenizer 对每个文本样本进行编码，然后通过自定义的数据集类 InputDataSet，将这些编码后的数据加载为 BERT 模型的输入。在该过程中，input_ids、token_type_ids 和 attention_mask 被构建并传递给 BERT 模型，实现了对文本数据的有效处理和建模。通过这种方法，可以方便地将大规模的文本数据转换为适用于 BERT 模型的输入格式，从而进行各种自然语言处理任务，如分类、问答和下一句预测等。如下图 1 为输入处理流程图。

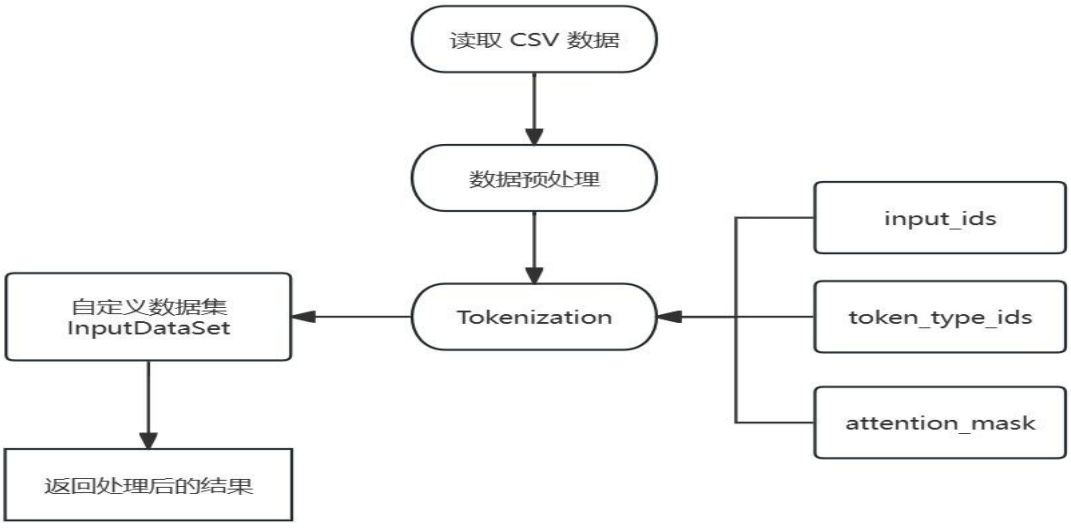

图 4-1 输入处理流程图

Figure 4-1 Enter a process flow chart

上述方法不仅简化了数据预处理的步骤，还提高了模型训练的效率和效果。如下代码块即模型的读入的数据的处理方式。

```
class InputDataSet(Dataset):
    def __init__(self,data,tokenizer,max_len):
        self.data = data
        self.tokenizer = tokenizer
        self.max_len = max_len
    def __len__(self,):
        return len(self.data)
    def __getitem__(self, item):   # item 是索引 用来取数据
        text = str(self.data['text'][item])
        labels = self.data['分类标签'][item]
        position_ids = self.data['专利名称'][item]
        labels = torch.tensor(labels, dtype=torch.long)#int-->long
```

其中__init__()里的 data 是 csv 文件内的数据，tokenizer 为分词器，max_len 为数据长度，即对数据的初始化函数。在__gititem__()里通过 item 索引取每条数据，每条数据进行提取，为后面的 embedding 处理做准备。

项目自主构建对文本进行封装，设计 fill_paddings 类进行补全句长，之后根据前面 Bert 模型的三个封装要求进行封装。

```
def fill_paddings(data, maxlen):
    '''补全句长，缺的补 0'''
    if len(data) < maxlen:
        pad_len = maxlen-len(data)
        paddings = [0 for _ in range(pad_len)]
        data = torch.tensor(data + paddings)
    else:
        data = torch.tensor(data[:maxlen])
    return data
# 手动构建
tokens = self.tokenizer.tokenize(text)
tokens_ids = self.tokenizer.convert_tokens_to_ids(tokens)
tokens_ids = [101] + tokens_ids + [102]
input_ids = fill_paddings(tokens_ids,self.max_len)
attention_mask = [1 for _ in range(len(tokens_ids))]
attention_mask = fill_paddings(attention_mask,self.max_len)
```

```
token_type_ids = [0 for _ in range(len(tokens_ids))]
token_type_ids = fill_paddings(token_type_ids,self.max_len)
```

构建完成后，返回结果有文本数据、input_ids、位置标记、分句标记、标签数据，中间三者对应前文所述的 token embedding、position embedding、segment embedding，如下图 4-2 展示的返回值。

```
return {
    'text': text,#文本数据
    'input_ids': input_ids,#vocab数字标记
    'attention_mask': attention_mask,#全1标记
    'token_type_ids': token_type_ids,#全0标记
    'labels': labels, #要输出的标签
}
```

图 4-2 返回值

Figure 4-2 Return Value

4.2 模型构建

项目用到 Bert 的预训练模型[10]BertPreTrainedModel，所有 Bert-based 的模型，包括预训练模型和下游任务模型都是基于 BertPreTrainedModel 类，用于初始化权重参数和加载预训练描述。

项目是要做文本分类，在模型构建中，实际上是对模型的微调[11]，首先需要提取出文本的特征，然后接全连接层，最后接一个 CrossEntropyLoss() 损失函数。

在文本分类任务里，pooled_output 通常作为模型的最终输出，被输入到一个分类器中进行分类。由于 pooled_output 包含了整个输入序列的语义信息，因此可以作为一个很好的特征表示输入序列，帮助模型进行分类，将这个特征作为全连接层的输入即可。代码里面还定义了 dropout 层，用于输出层，以减少模型在训练集上的过拟合，增强模型的泛化能力，在训练过程中有助于提高模型的鲁棒性和泛化能力，是 BERT 模型中的重要组成部分。

模型构建的整个流程如下，首先从数据源中获取文本数据，这些数据可能包含了需要处理的文本信息。接下来，通过分词器对输入文本进行分词处理，将文本转换为词片段，这样就得到了模型的输入数据。然后，将分词后的文本数据转换为 BERT 模型需要的输入格式，这个过程包括了 token embedding、segment embedding 和 position embedding 的处理，以便将文本信息转换为模型可以理解的格式。将嵌入处理后的数据输入到 BERT 模型中进行处理，BERT 模型将对输入的文本数据进行深层次的处理和理解。最后，获取 BERT 模型的输出结果，这些结果可能包括 logits（用于分类任务）和损失（如果提供了标签）等信息，可以用于后续的任务处理或

者模型评估。整个流程是一个端到端的自然语言处理流程，将原始文本数据转换为模型可以理解的格式，并获取模型处理后的输出结果。如下图 4-3 是模型构建的流程图。

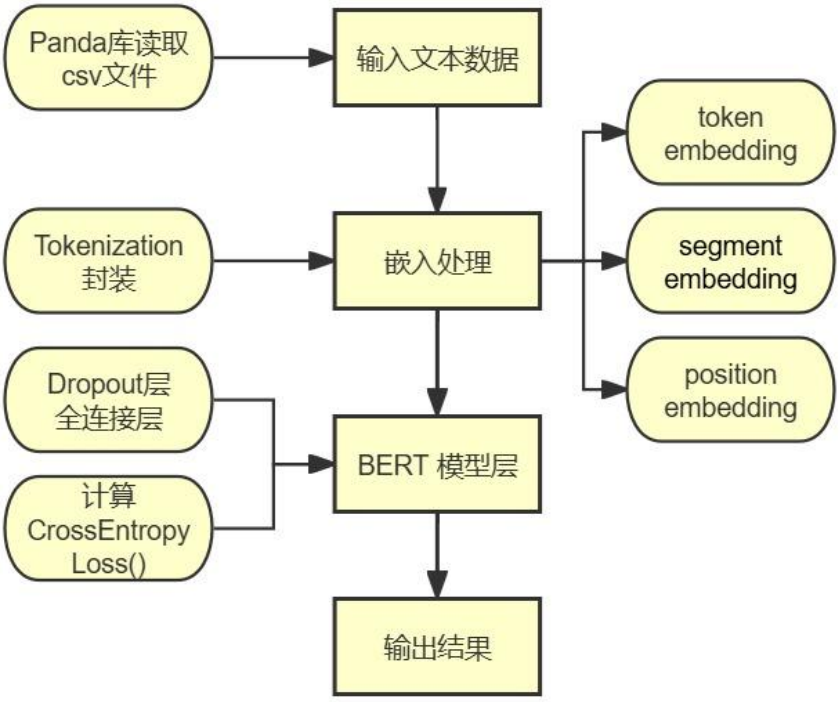

图 4-3 模型构建

Figure 4-3 Model building

模型的核心部分包括初始化函数和前向传播函数。在初始化函数中，模型参数被设置，包括标签数量、Bert 模型、dropout 层和分类器，并初始化了模型权重。前向传播函数输入经过 Bert 模型处理后生成输出，然后通过 dropout 层和线性分类器获得 logits。如果有提供标签，则计算预测 logits 与实际标签之间的交叉熵损失。代码如下。

```
class BertForSeq(BertPreTrainedModel):
    def __init__(self,config):   ##   config.json
        super(BertForSeq,self).__init__(config)
        self.num_labels = 124 #  类别数目,标签数目 A-H 大类的 124 个项目
        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, self.num_labels)
        self.init_weights()
    def forward(self, input_ids=None,   # 输入的 id,模型会帮你把 id 转成 embedding
```

```
                attention_mask=None,    # attention 里的 mask
                token_type_ids=None,    #[CLS]A[SEP]B[SEP]
                position_ids=None,    # 专利号位置 id
                labels = None, #  做分类时需要的 label
                return_dict = None
                ):
        outputs = self.bert(input_ids, #  文本数据依照 vocab 的 id
                                attention_mask=attention_mask, #  全 1 标记
                                token_type_ids=token_type_ids, #  全 0 标记
                                position_ids=position_ids, #  专利名称
                                return_dict=return_dict       #  预测值编号 A-H
                                )
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        loss = None
        if labels is not None:
            criterion = nn.CrossEntropyLoss()
            loss = criterion(logits.view(-1, self.num_labels), labels.view(-1))
        if not return_dict:
            output = (logits,) + outputs[2:]
            return ((loss,) + output) if loss is not None else output
        return SequenceClassifierOutput(
            loss=loss,    ##损失
            logits=logits,    #softmax 层的输入
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

在上面的代码中，定义了一个名为 BertForSeq 的神经网络模型，它继承自 Bert 预训练模型中的 BertPreTrainedModel。我们在 __init__() 方法即初始化函数，其包括 BERT 模型的初始化、设置输出标签数量、添加 Dropout 层以减少过拟合、定义线性分类器将隐藏状态映射到输出标签空间，以及定义交叉熵损失函数用于训练期间的损失计算。

在 forward() 方法即模型的前向传播过程，接受输入的文本 id、注意力 mask、

分句 id、标签和是否返回字典形式的预测值。它通过 BERT 模型获取输出，然后通过 dropout 层和线性分类器产生 logits，接着计算损失值（如果标签不为空）。最后，根据是否返回字典形式的预测值，返回损失值和 logits 或以字典形式返回损失值、logits 以及中间隐藏状态和注意力。

4.3 模型训练和评估

4.3.1 模型训练

首先，我们加载了预训练的中文 BERT 模型和相关配置，选择了 bert-base-chinese 作为我们的预训练模型。接着，我们利用 BertTokenizer 进行文本的 tokenization 和 embedding 初始化，以便进行后续的处理。

随后，通过已经准备了的训练数据和验证数据，使用 DataLoader 对其进行封装，以便进行批处理。在处理好数据之后，定义了优化器（AdamW）和学习率调度器，为微调过程做好了准备。然后，我们开始了训练循环，其中包括多个 epoch。在每个 epoch 中，模型设置在训练模式下，对每个 batch 进行前向传播、计算损失、反向传播和参数更新[13]。在每个 epoch 结束时，计算并打印训练集的平均损失，并对验证集进行评估，计算验证损失和验证准确率，并在控制台输出相应数据。最后保存了训练好的模型，以便于后续使用。以下是训练模块代码。

```
for epoch in range(EPOCHS):
    # 定义两个变量，用于存储训练集的准确率和损失
    total_train_loss = 0
    t0 = time.time()
    model.to(device)# 如果要进 gpu 运行，模型一定和数据位置一致
    model.train()
    for step, batch in enumerate(train_dataloader):
        optimizer.zero_grad() # 清除掉之前 batch 的 gradients
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        token_type_ids = batch['token_type_ids'].to(device)
        labels = batch['labels'].to(device)
        outputs=model(input_ids,
                      attention_mask=attention_mask,
                      token_type_ids=token_type_ids,
                      labels=labels)# 通过模型得到输出
        loss = outputs.loss   # 计算损失
        # 更新模型和调度器参数
```

```
            loss.backward()
            # 在 backward 得到梯度之后，step ()更新之前，使用梯度剪裁。
            nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()
            total_train_loss += loss.item()
            if (step + 1) % 100 == 0:
            scheduler.step()
        avg_train_loss = total_train_loss / len(train_dataloader) # 求平均损失
        print("avg_train_loss=",avg_train_loss)
        avg_val_loss, avg_val_acc = evaluate(model, val_dataloader)# 通过函数 evaluate 得到要验证数据的损失和准确率，这里验证数据=初始数据
        print("avg_val_loss=", avg_val_loss)
        print("avg_val_acc=", avg_val_acc)
```

训练完成后对模型进行保存，使用 torch.save()方法，方便后续加载直接使用模型，代码如下。

```
if epoch == EPOCHS-1:
    torch.save(model,'./cache/model.bin')
    print('Model Saved!')
```

4.3.2 模型评估

评估函数，用于在验证数据集上评估模型的性能，实现步骤如下，首先将模型设置为评估模式。在评估模式下，模型会关闭 Dropout 层和批量归一化层的功能，以确保在推断时模型的行为是确定性的。初始化验证集上的总损失，用于计算平均验证损失，迭代验证数据器中的每个批次，将输入的数据都移到设备上，项目用的是 GPU。之后在禁止梯度计算下，将输入传递给模型，得到模型的输出的预测结果、真实标签和损失。最后计算当前批次的准确率和当前批次的损失。所有批次结束后计算平均验证损失和平均验证准确率并将二者返回输出[14]。代码如下。

```
def evaluate(model, val_dataloader):
    model.eval()
    total_val_loss = 0
    corrects = []
    for step, batch in enumerate(val_dataloader): #每个批次分配一个索引 step。然后可以在循环中使用 step 来跟踪当前批次的索引
        input_ids = batch['input_ids'].to(device)
```

```
        attention_mask = batch['attention_mask'].to(device)
        token_type_ids = batch['token_type_ids'].to(device)
        labels = batch['labels'].to(device)
        position_ids = batch['position_ids']
        # print(position_ids)
        with torch.no_grad():
        outputs = model(input_ids,
                        attention_mask=attention_mask,
                        token_type_ids=token_type_ids, labels=labels)
        logits = torch.argmax(outputs.logits, dim=1)
        preds = logits.detach().cpu().numpy()
        labels_ids = labels.to('cpu').numpy()
        corrects.append((preds == labels_ids).mean())
        loss = outputs.loss
        total_val_loss += loss.item()
    avg_val_loss = total_val_loss / len(val_dataloader)
    avg_val_acc = np.mean(corrects)
    return avg_val_loss, avg_val_acc
```

5. 实验结果与分析

5.1 实验配置

在本研究中，我们进行了一系列实验，评估了我们提出的模型在不同环境下的性能。为了确保实验的可复现性和结果的可比性，我们记录了实验所需的硬件和软件环境，以及其他依赖项。

硬件配置：项目使用了一台配备 NVIDIA GeForce RTX 3050 Ti Laptop GPU 的个人计算机进行运算。该 GPU 提供了充足的计算资源，能够满足我们的实验需求。

软件环境：

操作系统：使用 Windows 11 操作系统作为实验的运行环境。

Python 版本：使用 Python 3.11.5 作为主要的编程语言和实验环境。

PyTorch 版本：使用 PyTorch 2.2.1 深度学习框架来实现和训练我们的模型。

CUDA 版本：使用 PyTorch 版本支持 CUDA 12.1，以利用 GPU 进行加速计算。

其他依赖项：项目使用了 NumPy 1.24.3、Transformers 4.32.1、torch 2.2.1 和 pandas 2.0.3 等库来辅助实验工作。

通过以上硬件和软件环境的配置，我们能够充分评估我们提出的模型在各种条件下的性能，并为进一步的研究和应用提供了重要参考。

5.2 实验结果

专利 IPC 编码的前一位是该专利的部，前三位是该专利的“大类”。本设计一共进行三种分类，一是对部的分类，二是对单独的 H 部大类的分类，三是对所有部的大类分类，通过观察三者的训练效果，能更加直观地表现 Bert 模型的效应。总体结果如下表。

表 5-1 实验结果

Table 5-1 Experimental results

类别	分类数目	平均验证集损失	平均预测集损失	平均预测准确率
A-H 部	5	0.30740	0.17284	0.96572
H 大类	8	0.29987	0.19948	0.96588
A-H 大类	124	1.44572	1.33654	0.69252

以上结果基于训练批次都为 4 情况，分类数目少情况下得到的具体数据优秀，分类数目多的情况下损失偏高。

5.2.1 对专利 IPC 部的分类

专利 IPC 编码的首字母是该专利的“部”，字母是 A-H，可分为 8 个类别,所用训练数据和测试数据比为 5 : 2。分类中设置实验批次（epochs）为 4、学习率（learning

rate）为2e-5、批次大小（batch_size）是4。

结果如下图5-1：

```
[line: 99] ==>
[line: 92] ==> ====Epoch:[4/4] avg_train_loss=0.30740====
[line: 93] ==> ====Training epoch took: 4:40:29====
[line: 94] ==> Running Validation...
[line: 97] ==> ====Epoch:[4/4] avg_val_loss=0.17284 avg_val_acc=0.96572====
[line: 98] ==> ====Validation epoch took: 5:08:41====
[line: 99] ==>
[line: 105] ==>
[line: 106] ==>    Training Completed!
```

图5-1IPC部的分类

Figure 5-1 Classification of IPC Part

上图可见，模型只经过4个批次训练。在训练阶段，平均训练损失为0.30740；在测试阶段，平均预测损失为0.17284，平均准确率为0.96572。综合来看，模型在训练数据和测试数据上表现良好，具有较低的损失和较高的准确率，这表明模型在训练过程中有效地学习到了数据的模式，并且具有较好的泛化能力。

5.2.2 对H部大类的分类

以H部的前三位为分类对象，可分为5个类别，从H01-H05，所用训练数据和测试数据比为1 : 1。分类中设置实验批次（epochs）为4、学习率（learning rate）为2e-5、批次大小（batch_size）是2。

结果如下图5-2：

```
[line: 97] ==> ====Epoch:[3/4] avg_train_loss=0.46089====
[line: 98] ==> ====Training epoch took: 2:24:31====
[line: 99] ==> Running Validation...
[line: 102] ==> ====Epoch:[3/4] avg_val_loss=0.30346 avg_val_acc=0.94512====
[line: 103] ==> ====Validation epoch took: 3:05:02====
[line: 104] ==>
[line: 97] ==> ====Epoch:[4/4] avg_train_loss=0.29987====
[line: 98] ==> ====Training epoch took: 2:22:00====
[line: 99] ==> Running Validation...
[line: 102] ==> ====Epoch:[4/4] avg_val_loss=0.19948 avg_val_acc=0.96588====
[line: 103] ==> ====Validation epoch took: 3:03:34====
[line: 104] ==>
[line: 110] ==>
[line: 111] ==>    Training Completed!
```

图5-2 H部大类

Figure 5-2 H-Class

上图可见只进行4个批次训练后的结果，在训练阶段，平均训练损失为0.29987；在测试阶段，平均预测损失为0.19948，平均准确率为0.96588。

5.2.3 对所有部大类的分类

项目以 A-H 部的前三位为分类对象，共 124 个类别，所用训练数据和测试数据比为 5 : 2。分类中设置实验批次（epochs）为 7、学习率（learning rate）为 2e-5、批次大小（batch_size）是 8。

结果如下图 5-3：

```
[line: 99] ==> ====Validation epoch took: 3:19:09====
[line: 100] ==>
[line: 93] ==> ====Epoch:[4/4] avg_train_loss=1.44572====
[line: 94] ==> ====Training epoch took: 2:47:13====
[line: 95] ==> Running Validation...
[line: 98] ==> ====Epoch:[4/4] avg_val_loss=1.33654 avg_val_acc=0.69252====
[line: 99] ==> ====Validation epoch took: 3:06:23====
[line: 100] ==>
[line: 106] ==>
[line: 107] ==>    Training Completed!
```

图 5-3 所有部大类

Figure 5-3 All major categories

上图可见只进行 4 个批次训练后的结果，在训练阶段，平均训练损失为 1.44572；在测试阶段，平均预测损失 1.33654，平均准确率为 0.69252。

5.3 小结和遇到的问题

总体的实现比较成功，通过上述三种分类方式的结果，分类从 5 到 8 到 124，分类的平均准确率分别是从 0.96588、0.9700 到 0.69252，用基本一致的分类方法，通过修改提供训练的数据和训练的参数得到结果，就是到实现模型的 124 个 IPC 分类时在训练批次比较少情况下导致损失还没收敛到 1 以下，但预测的准确率还是比较高的，可见运用 Bert 模型对专利 IPC 的分类效果是十分优秀的。

实验过程中，出现过由于数据量和参数问题导致模型训练过拟合[15]和欠拟合问题，数据模块，数据需要通过打乱，并且分为训练集和测试集。其中如果数据没进行打乱就训练，会导致模型过拟合，最大问题就是在训练中模型增加判断正确样本的置信度，导致训练中的 loss 会下降，但预测 acc 保持稳定。通过对实验数据的修改和调节学习率和批次，成功解决了这类问题。实验还遇到分类类别数目过大导致的损失过大，即前面的对所有部的分类，有一个修改方向，通过对每个部进行训练，即对 H 部类的分类那样，之后通过模型叠加实现 IPC 的分类，这样得到模型的精确度会更高，更好的实现专利 IPC 的分类。

6. 总结

本毕业设计旨在利用自然语言处理技术中的预训练模型 Bert（Bidirectional Encoder Representations from Transformers），结合专利文本数据，实现专利 IPC（International Patent Classification）的分类。IPC 是对专利文献进行分类的国际标准，对于专利检索和分析具有重要意义。通过将 Bert 模型应用于 IPC 分类任务，目的在提高分类的准确性和效率。设计主要用的语言是 Python，基于 Bert，充分利用了自然语言处理技术的最新成果，在专利文本分类任务上取得了一定的成果。通过对模型的优化和数据的不断丰富，相信能够进一步提高分类的准确性和效率，为专利检索与分析提供更好的技术支持。

参考文献

[1] 廖列法,勒孚刚,朱亚兰.LDA 模型在专利文本分类中的应用[J].现代情报,2017,37(03):35-39.

[2] 王晶.基于深度学习的文本表示和分类研究[D].北京邮电大学,2019.

[3] 赵阳,周龙,王迁,等.民汉稀缺资源神经机器翻译技术研究[J].江西师范大学学报(自然科学版),2019,43(06):630-637.DOI:10.16357/j.cnki.issn1000-5862.2019.06.12.

[4] 刘保臣. 基于多维信息融合的监狱风险评估和预警方法研究[D]. 山东大学,2022.DOI:10.27272/d.cnki.gshdu.2022.001795.

[5] 柏政含,何欣楠,徐映千.基于多头注意力机制下的关键词预测模型——以南京大学图书馆为例[J].信息与电脑(理论版),2022,34(22):202-205+222.

[6] 姚凌峰. 基于深度学习识别医学文本中的 PICO 成分[D]. 南京邮电大学,2021.DOI:10.27251/d.cnki.gnjdc.2021.001009.

[7] 朱群雄,孙锋.RNN 神经网络的应用研究[J].北京化工大学学报(自然科学版),1998,(01):88-92.

[8] 黄可鸣.关于机器学习[J].计算机科学,1987,(01):29-32.

[9] 杨飘,董文永.基于 BERT 嵌入的中文命名实体识别方法[J]. 计算机工程,2020,46(04):40-45+52.DOI:10.19678/j.issn.1000-3428.0054272.

[10] 白子薇.基于预训练语言模型的机器阅读理解技术研究[D]. 北京邮电大学,2022.DOI:10.26969/d.cnki.gbydu.2022.000277.

[11] 邸月.基于 BERT 的中文专利分类方法研究[D]. 河北工业大学,2019.DOI:10.27105/d.cnki.ghbgu.2019.000682.

[12] 高德亮.基于 AdamW 算法的 WT-GRU 模型在高频股指预测中的应用[D]. 山东大学,2021.DOI:10.27272/d.cnki.gshdu.2021.004954.

[13] 王懿.基于自然语言处理和机器学习的文本分类及其应用研究[D].中国科学院研究生院（成都计算机应用研究所）,2006.

[14] 杨中成.融合预训练语言模型的机器译文质量评估[D]. 北京交通大学,2019.DOI:10.26944/d.cnki.gbfju.2019.001474.

[15] 陶砾,杨朔,杨威.深度学习的模型搭建及过拟合问题的研究[J]. 计算机时代,2018,(02):14-17+21.DOI:10.16644/j.cnki.cn33-1094/tp.2018.02.004.

致谢

本分类设计作为大学的总结，运用到大学知识，通过自主学习实现并完成，在实现中感谢百度和 CSDN 的相关人员的知识分享，给我实现本课题提供了帮助。同时感谢老师和师兄的指导，对实践报告的改进提出宝贵的建议，而且还在我遇到困难时尽心地进行指点与解答。在此借毕业实践报告完成之际，表示由衷的感谢和敬意。