在本次项目中，我承担的任务是后台的编写，cnn模型的构建和html响应函数的编写。

其中后台部分，我们采用了node.js作为后端服务器。因为我们对后台部分的要求不高，只需要用户能够上传训练数据，所以我们采用了node.js这种较为简单的方式。而在编写后台代码的过程中，我还是遇到了不少的难题。第一个便是静态文件的存储位置。在这之前我没有过后台代码的编写经历，完全没有意识到在服务器端读取html,css等静态文件的路径问题。在web学习的课程中，我们只需要在html,js等文件中输入文件在我们电脑的绝对路径就能访问到该文件，而在构建本地服务器的时候，明显不能通过这种方式来访问文件。这个不大不小的问题是我编写后台代码的第一个难点，因为node.js作为本地服务器的使用人数较少，大多使用Java作为后台代码的编写语言，所以较少有人编写关于node.js的使用教程的博客，阅读官方的api文档则难以快速上手。第二个问题则是相类似的路径问题，因为上传文件后会默认保存在C盘，我需要指定其存放位置。第三个问题是我需要通知前端文件已经上传完毕。这三个问题都算是比较常规的后台的编写需要学习的过程。

对于cnn模型的构建，难点则在csv文件的读取部分。在mlp模型中，因为只有一个输入和一个输出，数据量较小，所以我们选择了一同读取。而在cnn中，训练的是图片，而且csv格式便注定了图片的像素点的值需要平铺开，那么一个图片需要依次读入上百个像素点，这是还是mnist数据集的图片比较小。因而不能像mlp那样一同读取所有的图片，那样的话浏览器的数据缓存压力很大，我们需要批量读进来，每次读一个batch。对于csv文件的读取，tfjs有专门的api可以读取。我学习了官方的api文档，但是依然会出现一些小问题。所以我转而使用了paparse.js这个库来读取csv文件。最后成功根据batch的大小返回了数据。

在此之后我尝试运行了我们的模型，对于网络层的构建我们虽然没有像googlennet,alexnet那么庞大，但是应该能够很好的拟合mnist这个比较小的数据集。可是我们的结果却不甚理想。后来我想了想，应该是缺少了归一化，于是我对数据进行了归一化操作，损失值很快便降了下来。

再有一个就是Html响应函数的编写。这个主要是是指不正确的输入，错误提示等。包括了文件没有正确上传，超参数设置成了字符串。还有文件选择后显示文件路径等。这些都是比较简单js响应函数。

这次项目我比较全面的参与到了前端和后台的代码编写过程当中。尤其是对后台的代码，有了比以往更见全面的认识。对于前端和后台的交互，也有了一定的了解。学习了node.js在作为本地服务器的时候，如何设置静态文件地址，如何上传文件，以及在文件接收完成后向前端发回信息。而对于cnn模型的构建，学习的更多的是对于csv文件在js中的读取。对于cnn模型本身，我在python中便有过许多次使用，也非常的熟悉。起初我们是没有确定哪种文件类型作为训练数据的，然后我去了解许多的官方实例，发现大部分都是csv文件，然后我们也使用了csv文件作为训练数据来使用。csv文件在JavaScript中也确实有非常多的读取方式，仅是我在这次项目中就学习到了tfjs,papa,d3等三种方式。在这其中，tfjs是花费了我最多的时间去学习的。但是很令我失望的是我并没有熟练的学习到其使用方式，只能无奈放弃。这其中应该有很大的原因是我本身对tensorflow的使用不是很熟练，在tensor向量和data之间的转换非常不熟悉。但是也因此我也学习了d3和papa另外两种读取方式。这个项目本身而言还差一点，我们起初是想要完成mlp,cnn和rnn三种模型的。而且我认为最大的难度—数据读取已经被克服了。但是我还是小瞧了一个神经网络，对于rnn神经网络实在不甚熟悉。调试的过程中出现的问题不知如何解决，便放弃了rnn。另外一点不足是，我们的读取来的数据还原成图片的时候不是很成功。其实原本想把像素点重新排列成矩阵展示给用户观看的。但是对于不同的训练数据好像确实不能左到如实还原成图片。

总的来说，这次项目令我收获颇丰，也是到目前为止我安排时间上最为合理的一次课设。没有赶工，早早便完成了我的工作，给自己留了许多余地，以便出错了能及时修改。