Using MSRN to extract Oncological Outcomes from Radiology Reports

Malcolm Meyerson, Singh Saluja, Alexandra Rodríguez, Ved Narayan
https://github.com/arrowandbead/RadiologyReportNLPAnalysis

Introduction

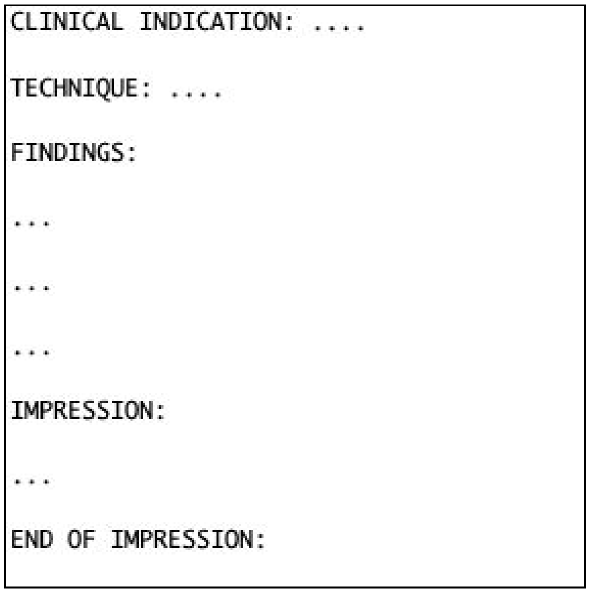
Machine learning can be a catalyst for rapid health care systems. In these systems, patient data can be used in real time to carry out cost-effective, personalized and low-risk treatments. Our project tackles the problem of finding a scalable method to extract clinical endpoints from electronic health records or more specifically, oncological radiology reports. To this end we collaborated with Brown's Radiology AI Lab. They provided us with all data and labels we used for this project. Usually, reports come in large amounts and it is sometimes difficult to gauge the meaning of the author's impression, as it can include a lot of technical language. Our goal is to build a natural language processing model capable of describing a patient's disease progression and response to therapy from their radiologic reports. To accomplish this, we adapted various BioBERT models capable of describing whether a radiology report signals there is evidence of cancer [2][3]. The result is outputted as a numerical label, indicating varying degrees of cancer evidence (as seen in the table below).

Label	Definition
1	*Yes, the report states there is evidence of cancer*
2	*Yes, the report states or implies there is evidence of cancer*
3	*No, the report states there is no evidence of cancer*
4	*No, the report states or implies there is no evidence of cancer*
5	*The report mentions cancer but is uncertain, indeterminate, or equivocal*
6	*The report is uncertain, indeterminate, or equivocal*
7	*The report does not mention cancer*

Table 1: Evidence of Cancer Classifications

Data & Preprocessing

We had available a folder of 923 reports in the form of individual text files from 138 pediatric rhabdomyosarcoma patients and an Excel spreadsheet with each corresponding report's annotation (evidence of cancer label, 1-7). This data was anonymized and provided to us by the Radiology AI Lab and is not publicly available. Each report in general was a text file similar to Figure 1, including more sections and information about the patient, order, etc.



```
CLINICAL INDICATION: ....

TECHNIQUE: ....

FINDINGS:

...

...

...

IMPRESSION:

...

END OF IMPRESSION:
```

Figure 1: Sample Report

In our preprocessing, we carefully extracted each impression from the reports, by looking for the "IMPRESSION" section of each and accounting for edge cases and inconsistencies, such as impression sections appearing twice in one report and reports ending with "END OF IMPRESSION", as above. While doing this, we compiled a dictionary of reports to impressions by parsing every report accession id from its file name and mapping it to its impression section (a string). We then extracted the label of each report from the Excel file, while also making a dictionary of every report id to corresponding evidence of cancer label. Once we gathered these two dictionaries, we constructed an array of impressions and an array of labels aligned to correspond to each other. The challenges in this were finding and correcting inconsistencies in report ids, such as unicode commas in the report id strings coming from the excel file, and excluding reports that were not labeled. After preprocessing, we arrived at 908 labeled examples from the original 923. Furthermore, for each class, we found we had a distribution of examples as depicted in Table 2.

Class	1	2	3	4	5	6	7
# of examples	248	72	363	127	77	25	3

Table 2: Distribution of Examples Per Class

Model Architecture

Our implementation uses BioBERT, a model with a similar architecture to Google's BERT that is pre-trained on files with biomedical language. With BioBERT, we tokenize our input into sequences of length 128 and extract attention masks. We then use BioBERT again as the first layer of our model to grab the embeddings of our impressions. After we have these embeddings, we pass them through global max pool, batch normalization, a dense layer (with hidden size equal to 128 and a rectified linear activation function or ReLU), a dropout layer with a rate of 0.1 and a final dense layer of size seven with a softmax activation function. Our model uses an Adam optimizer and Categorical Cross-Entropy loss because of the nature of our classification task. We trained our model on an 80-20 train-test split with the hyperparameters as defined in Table 3 below.

learning rate	0.001
batch size	32
Number of epochs	20
Activation functions	ReLU & Softmax
Optimizer	Adam
Dropout	0.1

Table 3: Hyperparameters

Our architecture is as depicted in Figure 2, where 32 pertains to the batch size, 128 to our sequence length and 768 to the number of hidden states in BioBERT.

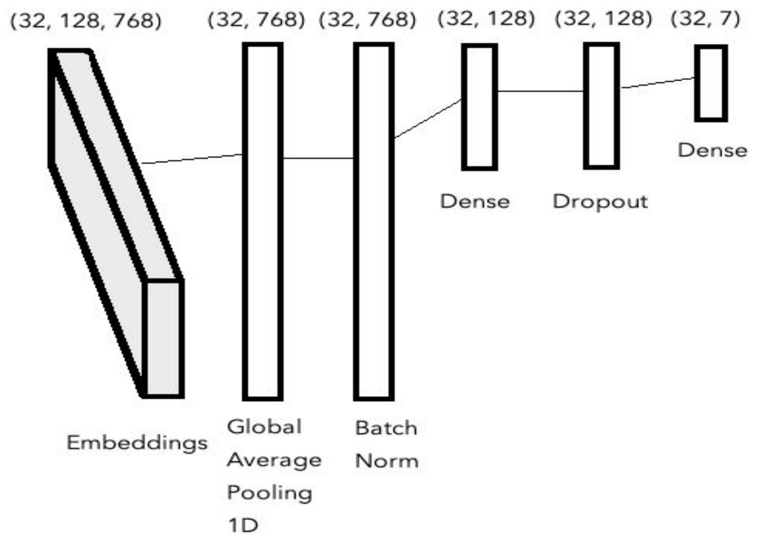


Figure 2: Model Architecture

Even though we had scarce data, we found that this model and hyperparameters optimized for a balance of relatively high accuracy (around 75% overall) and relatively high per-class accuracies as well (max 94% for the most frequent label) that were similar to the distribution of per-class examples. We found that increasing the size of the hidden layer polarized per-class accuracies in that the gap between the accuracy of the scarcest label diminished substantially while the per-class accuracy of the most frequent labels increased dramatically (reaching 97%) while overall accuracy was low (61%). Similarly, our combination of batch normalization and low number of epochs worked to create a fast model, but our low learning rate, despite the low number of epochs, drove our model to reach the highest accuracies.

Results

Because our data had an unbalanced representation of each class, we added as a second metric the per-class accuracy. As we expected, the per-class accuracy of the most frequent labels was the highest. To calculate the accuracy of the model as a whole, we used Keras's Categorical Accuracy. Our best run had a train Categorical Accuracy of 72.75482% and a test accuracy of 72.7248%. These accuracies only vary by a very small margin, which we think could be made more robust with a larger dataset. The accuracy for each class is as described in Table 3.

Class	1	2	3	4	5	6	7
Accuracy	78.0%	30.0%	94.2%	44.4%	35.3%	0.0%	0%
# of examples in testing	54	10	69	27	17	5	0

Table 3: Per-class test accuracies

As we can see in the table above, the accuracies for the least represented classes werethe lowest and 0 in the case of 7, since there were no examples pertaining to the label in the test batch. Figure 3 shows the change in train accuracy across epochs.

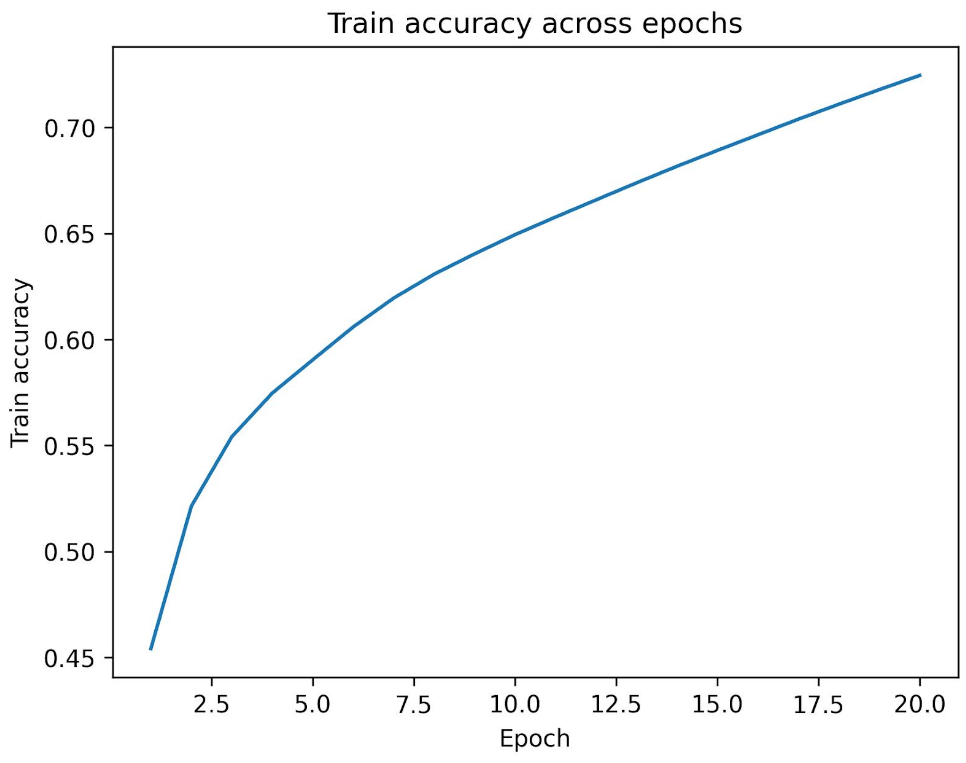


Figure 3: Train accuracy across epochs

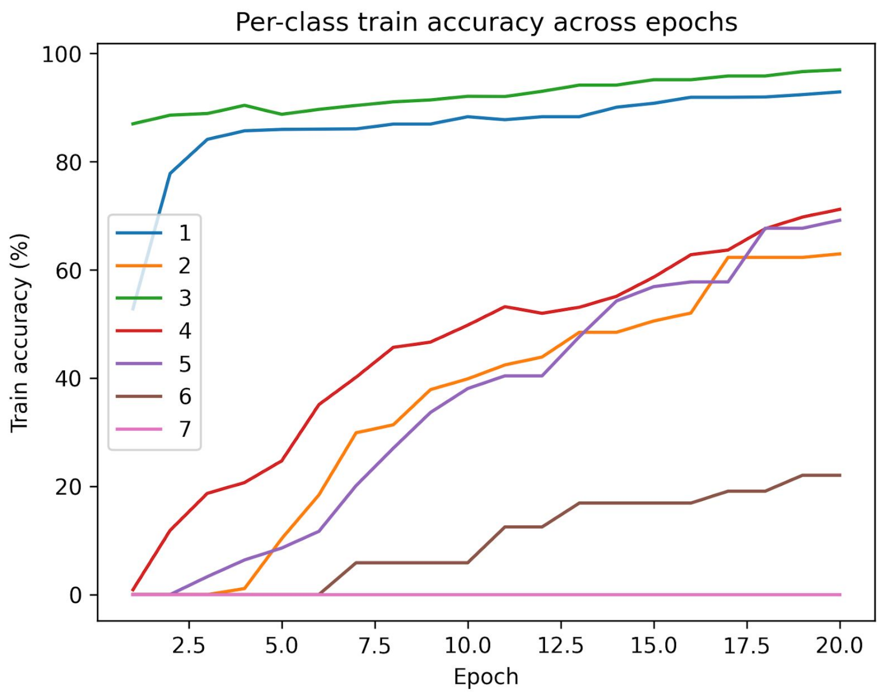


Figure 4: Per-class train accuracy across epochs (Full architecture)

Figure 4 shows the per-class train accuracy across epochs. We can see how these accuracies are greater than they are for our test dataset, yet they still align with the distribution of each class within our examples. The most frequent class, 3, has the highest train accuracy and the least frequent class, 7, has the lowest accuracy.

Furthermore, we ran a model that includes only global average pooling and a dense layer mapping the output to the seven labels. We constructed this architecture as a base model to compare to our model and depict its strengths. The base model achieved a train Categorical Accuracy of 60.95041% (compared to our original model's 72.75482%) and a test accuracy of 61.01211% (compared to our original model's 72.7248%). The train accuracy and per-class accuracies of the base model across epochs are depicted in Figures 5 and 6, respectively. We can conclude that our architecture provides more scare classes with higher accuracy as compared to the base model, as well as a higher accuracy overall.

Class	1	2	3	4	5	6	7
Accuracy (full architecture)	78.0%	30.0%	94.2%	44.4%	35.3%	0.0%	0%
Accuracy (base model)	87.0%	10.0%	89.9%	29.6%	11.8%	0.0%	0%

Table 4: Per-class test accuracies across both models

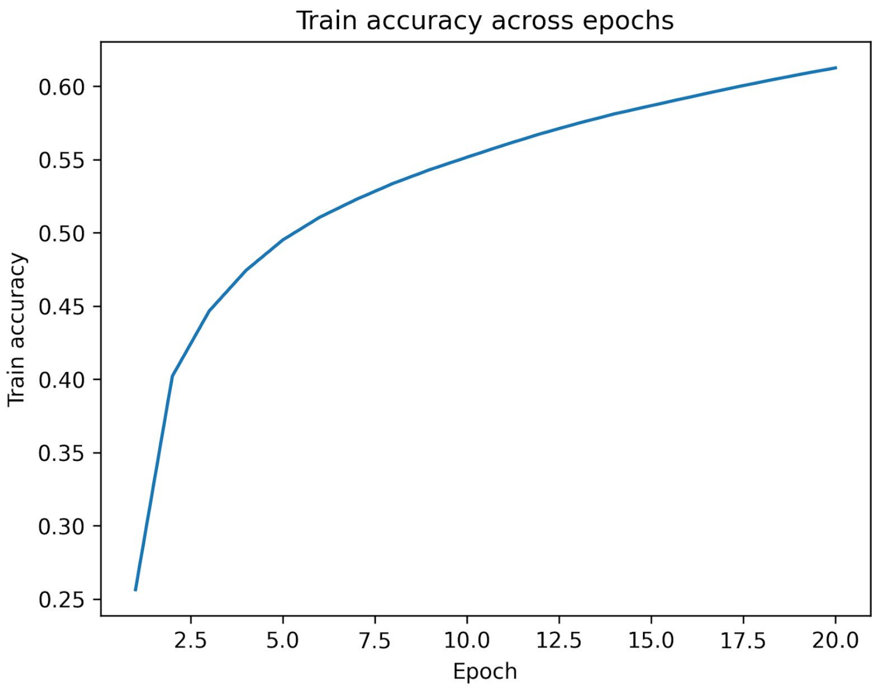


Figure 5: Train accuracy across epochs (base model)

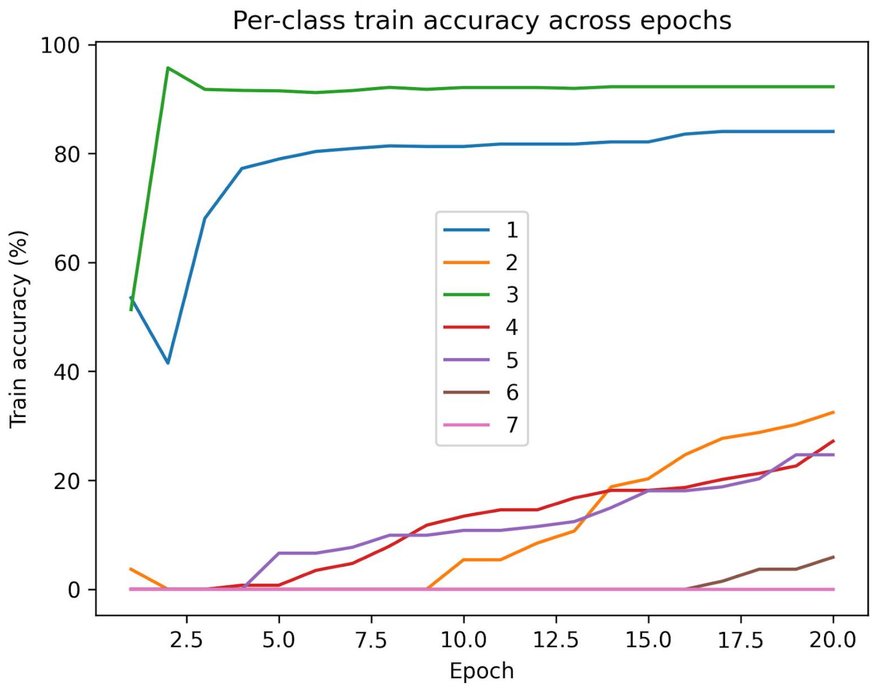


Figure 6: Per-class train accuracy across epochs (base model)

Challenges

One big challenge of our project was the scarcity of annotated data available to us. With just 908 examples, we struggled to find a model and a run that would reflect meaningful results. Moreover, at the start of our implementation, we programmed a model inspired by an article that tackled a similar classification task, but were not able to explain the high accuracy the model was outputting. We shifted from using Keras's API functions such as fit() and evaluate() to writing our own call, train and test functions. In addition to this, we examined the robustness of our model by calculating the per-class accuracies, which reflected the imbalance in our data.

Furthermore, another challenge we faced was that the model quickly outgrew the computational capacity of our local workstations. It takes about more than an hour to train. We leveraged Oscar (Brown's Supercomputer). On a single node with 32 cores it takes about 30 minutes to train. But as the model became complex the CPU processing pipeline was not efficient enough. We then switched to GPUs, we mostly trained on a single Quadro RTX 600 (24GB). On a GPU, it takes 3 minutes to train.

Discussion & Future Work

This project was multifaceted and presented many challenges along the way. Firstly, parsing the inconsistently sectioned data was a team effort that required multiple iterations. There were many rare reports in the dataset that denoted the beginnings and endings of their sections in uncommon and sometimes singular ways. We then had to come up with a plan to work with low amounts of data for a task that traditionally requires large amounts of data (NLP). Finally, our initial models (using the tf.keras.Model framework) produced high accuracy numbers (sometimes 99% for previously unseen data), which we could not explain. This in particular required long periods of group discussion and problem solving. The large disparity in the number of data points also made the overall data set accuracy less useful for determining the efficiency of our model. Ultimately we programmed our own accuracy functions to sanity-check our model and expanded upon our metrics to include per-class accuracy. As mentioned in our challenges section, we shifted our implementation and instead of using Keras's functions to train and test our model, we wrote our own call(), train() and test() functions. With these, we were able to produce accuracies that were both reasonable and relatively good for classes where we had the most data. Classes 1 and 3 in particular were correctly classified with high accuracy.

As it stands our model is not ready for the original concept of triaging patients. We did not have enough data to produce a model that could accurately predict the rarer classes in the data set. This means that if our model was evaluating thousands of reports every day, many patients would be misclassified leading to adverse health outcomes. The most direct response to this problem would be to train with more data. Annotating reports, however, is a time-consuming task and the lab we were working with had only a couple of med students and an MD to label radiology reports by hand. A larger amount of data (10x perhaps) would probably allow us to achieve similar results to those of the most common classes in our dataset.

Another prospect would be to use weakly labeled data sets. For example, if we got per class accuracy to reasonable levels for the most important classes we could then use the model to label thousands of more reports and then learn off of the model-labeled reports. Weakly labeled data is still a new frontier in deep learning, but it is a promising alternative to resource intensive hand labeling of data sets. The approach proposed above would still require more ground truth hand labeled data than we currently have.

Finally, our model could be adapted to meet the Lab's second objective, which is to extract the progression of the disease, designated in similar labels (a through e), for improvement, worsening or uncertainty as expressed by the author of the report.

From an ethics perspective we were working with real patient data so we took extra care to avoid uploading any data to GitHub. This was ensured by only ever storing our data inside a folder which was git-ignored. The Radiology AI Lab we collaborated with anonymized the data prior to giving it to us, complying with HIPAA regulations. As I stated previously our model is not ready to be used on actual patient data in clinical contexts.

Reflection

Constructing this model was a difficult task and we learned a lot along the way. Though at first we worked on multiple approaches (an architecture that used Keras API functions, a PyTorch approach and our own Transformers) we ultimately implemented our model in the same way as we implement our homeworks. We struggled from the start with understanding the problem, narrowing down achievable goals and testing the robustness of the results our implementations were returning. However, we were able to collaborate successfully by providing each other with resources to run the models (such as setting up CCV accounts), writing code together (through VSCode liveshares) and testing each other's models (by programming alternative accuracy functions).

Along the way, we were able to challenge our understanding of both the task at hand, our implementations and the metric of ultimate efficiency for our model. The project allowed us to practice thinking critically about why certain layers resulted in specific accuracy measurements and how to construct a model from scratch in a way that allowed to directly see everything that is going on (inside of call(), train() and test() and in the way in which our data is split). We also learned about the influence of data on model outcomes and were surprised to have reached the accuracies we did, despite our scarce resources.

At the end of the day, it was fun to collaborate and challenge each other to produce a robust model and prioritize our tasks. We especially enjoyed fiddling with our model's layers, hyperparameters and more and discovering along the way the weaknesses and strengths of our implementation. Moreover, it was a privilege to be able to collaborate directly with a lab at Brown. This project was certainly a great opportunity to lay the foundation for building models to solve tasks on our own. Overall, our team is satisfied with the results, enjoyed the entire process and now feel more confident in our skills!

Citations

[1]Devlin et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
[2] Lee et al. (2019). BioBERT: a pre-trained biomedical language representation model for biomedical text mining
[3] James Briggs. (2020). TensorFlow and Transformers
[4] Giacomo Miolo. (2020). Github Repository,
https://huggingface.co/giacomomiolo/biobert_reupload