Search, Retrieval, and Classification of AI-generated art prompts

Mohamed Darkaoui,
Viktor Hura,
Mounir Madmar

December 18, 2022

Contents

1 Introduction

The popularity of AI-generated[7] art is increasing, leading to a growing demand. By simply providing the AI system with a written description of the desired image, the system will generate one or more images based on that description. This has been made possible with models such as DALL-E[10], Stable Diffusion[12], Midjourney[11], ...

The purpose of this project is to provide users with the ability to view images that have been generated using similar prompts, in order to conserve computational resources and potentially inspire the user with previously generated images. By allowing users to retrieve these previously generated images, we hope to make it more accessible and efficient for everyone.

For retrieval we used Elasticsearch[15] which is a distributed, free and open search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. It is based on the Lucene[3] library.

In addition, we have used machine learning models to label our dataset[18] of 1.5 million prompts with 28 different fine-grained emotions[13], allowing the user to narrow down the prompts by the emotions that best fits their artistic vision.

2 Data

Lexica[20] is an AI-art search engine, a gallery for artwork, offering a text to image model capable of generating images given any text input. It gets its images from the official Stable Diffusion[12] Discord channel or generates them directly. It is accessible via API and has its own dump which contains the text prompts along with the links of the generated image inside one messy TXT file.

In order to improve the structure of our data, we have migrated the TXT file format to the CSV file format. This change allows us to add a separate column for each emotion, which enables us to perform emotion analysis on our data. We used the pandas library[24] together with the Go programming language[30] to accomplish this transformation.

For the purpose of training our emotion classification models, we make use of Google's GoEmotions dataset[13].
GoEmotions is a human-annotated dataset of 58k Reddit comments labeled with 27 emotion categories. To our knowledge, it is currently the largest English language fine-grained emotion dataset available to the public.
GoEmotions' taxonomy includes 12 positive, 11 negative, 4 ambiguous emotion categories and 1 "neutral", which made it very appealing for us in contrast to other datasets that we considered, which mostly contained 5 or so basic emotion categories[28].

Figure 1: GoEmotions taxonomy: 28 emotion categories, including "neutral".[13]

Positive		Negative		Ambiguous
admiration 👋	joy 😃	anger 😡	grief 😢	confusion 😕
amusement 😂	love ❤️	annoyance 😒	nervousness 😬	curiosity 🤔
approval 👍	optimism 🤞	disappointment	remorse 😔	realization 💡
caring 🤗	pride 😌	disapproval 👎	sadness 😞	surprise 😲
desire 😍	relief 😅	disgust 🤮		
excitement 🤩		embarrassment 😳		
gratitude 🙏		fear 😨		

Figure 2: Example annotations from GoEmotions [13]

Sample Text	Label(s)
OMG, yep!!! That is the final answer. Thank you so much!	gratitude, approval
I'm not even sure what it is, why do people hate it	confusion
Guilty of doing this tbph	remorse
This caught me off guard for real. I'm actually off my bed laughing	surprise, amusement
I tried to send this to a friend but [NAME] knocked it away.	disappointment

It goes without saying that emotion classification is a highly complicated and subjective task, which is very challenging. The dataset and the challenges are further discussed in the emotion classification section of this report.

3 Indexing

Elasticsearch[15] is based on the Lucene[3] library and uses the inverted index data structure to index the data. It stores data as JSON documents. When a document is added to an Elasticsearch index, the text of the document is analyzed and broken down into individual terms. Elasticsearch comes with a number of built-in analyzers that can be used out of the box, including the standard analyzer, the simple analyzer, the whitespace analyzer, and the stop analyzer.

The PyCharm Elasticsearch[2] plugin is utilized to import and convert the data into JSON format. Upon establishing the connection, the plugin can be utilized to index the data automatically using the standard analyzer, which tokenizes the text, eliminates stop words and lower-cases the text. We chose to use the standard analyzer because it is the most suitable for our project.

4 Search

The purpose of the whole project is to be able to retrieve data using queries. Elasticsearch uses a query language called Query DSL (Domain Specific Language) to define the query and its search criteria. Queries can be run directly from the Elasticsearch API, or they can be executed using a library in a programming language.

In our project, we utilize the Python Elasticsearch-DSL[16] library in order to achieve cleaner code. This library facilitates the interaction with Elasticsearch in Python, which allows us to focus on the logic of our search queries rather than the syntax of the Query DSL.

Elasticsearch supports two categories of queries: leaf queries and compound queries. Leaf queries are simple queries that match a specific value in a specific field, such as a keyword search. Compound queries are more complex queries that combine multiple leaf queries or other compound queries.

In our project, we mainly use the compound queries in order to search for specific text and filter unwanted results. To give a brief example, the user can search for prompts containing the word "clown" with anger level between 0.8 and 1. This type of query allows us to retrieve prompts that are related to clowns and have an angry connotation.

In Elasticsearch, the relevance of search results is automatically ranked based on relevance scores. The scores reflect how well result matches the search query. They are used to determine the order in which the results are presented to the user.

Our system supports fuzzy search and exact search. Fuzzy search allows for matching results that are similar to, but not necessarily exactly the same as the search term or phrase. In contrast, an exact search requires an exact match of the search term in order to return results.

5 Emotion Analysis

5.1 Goal

The goal of this section was to train a model on the GoEmotions dataset, such that we can apply said model on our Lexica AI prompts database[18]. This is a multi-label text classification problem, where we have 28 different emotion classes.
In addition, we created our own models and compared them to the reference models used in the GoEmotions paper[14].

For our own models we wanted to use Convolutional Neural Network(CNN)[9] based models, which are more commonly used in image classification tasks.
We have seen examples of such models being used in sentiment analysis, where word embeddings make it possible to treat such text data as spatial data, analogously to the way we treat image data in such models.[23, 21]

But these examples are usually limited to a small amount of labels (e.g. positive/neutral/negative), and only a single label is assigned to a given text sequence.

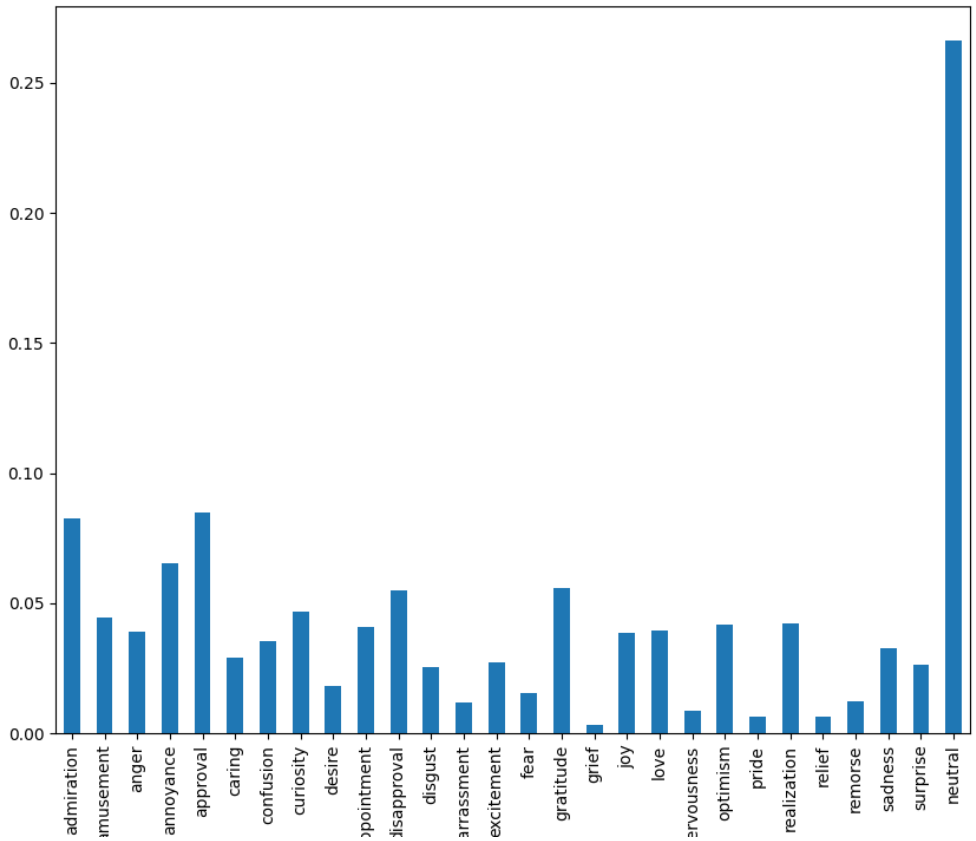
Applying these CNN based models on such a large multi-label classification problem, will show us just how far such models can take us, and demonstrate the effectiveness of domain specific models such as BERT[8] and PRADO[19].

5.2 Challenges

When working with the GoEmotions dataset, we have encountered many challenges and limitations, some of which were also highlighted in Google's paper[14].

On first inspection, we can see that distribution of labels is very unbalanced as seen in the figure below. Neutral emotions are over-represented and emotions such as grief, pride and relief, are under-represented. This unbalance may manifest itself in our classification results.

Figure 3: GoEmotions dataset label distribution.



Emotions are very complex and subjective, especially on such a granular level, and we are not sure that 58k internet comments would be enough to capture all that nuance.
In addition only 82 annotators were tasked with labelling this data, where only 3-5 annotators are used for each text sequence. And 83% of all sequences only have a single label.
Several other flaws and biases were highlighted in the paper itself and in this article from Surge [1].

For our project, it is important to acknowledge the limitations of our dataset, use the model metrics for comparison, and treat our classifications as more of a proof of concept for what might be possible with more advanced natural language processing models and emotion classification.

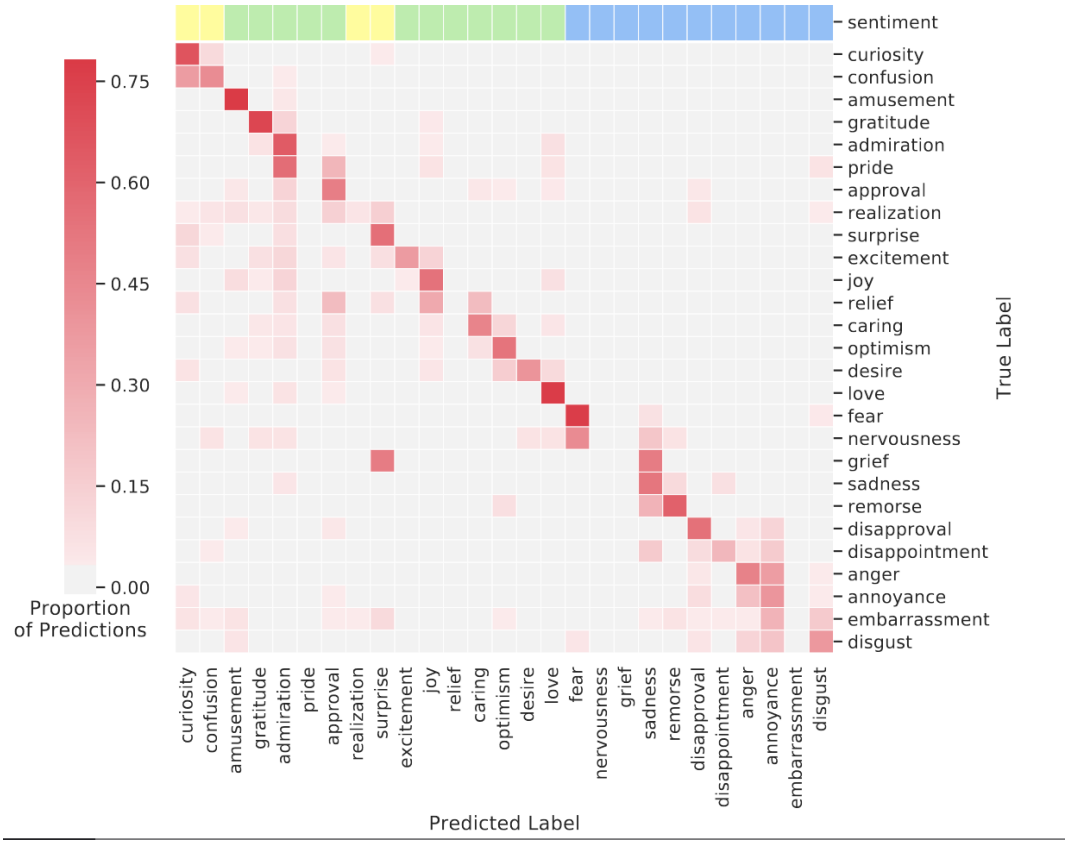
5.3 BERT reference

The researchers in the GoEmotions paper made use of transfer-learning to fine-tune a BERT[8] model on this dataset.
Their model achieved the following macro-average metric scores when applying their model on the validation set:

Precision	Recall	F1
0.40	0.63	0.46

And they gave the following confusion matrix for their predictions:

Figure 4: GoEmotions BERT confusion matrix. [8]



As we can see, there is a nice line forming along the main diagonal, which means it's guessing the right label most of the time, but the model confuses some emotions with other emotions that are related in intensity and sentiment.

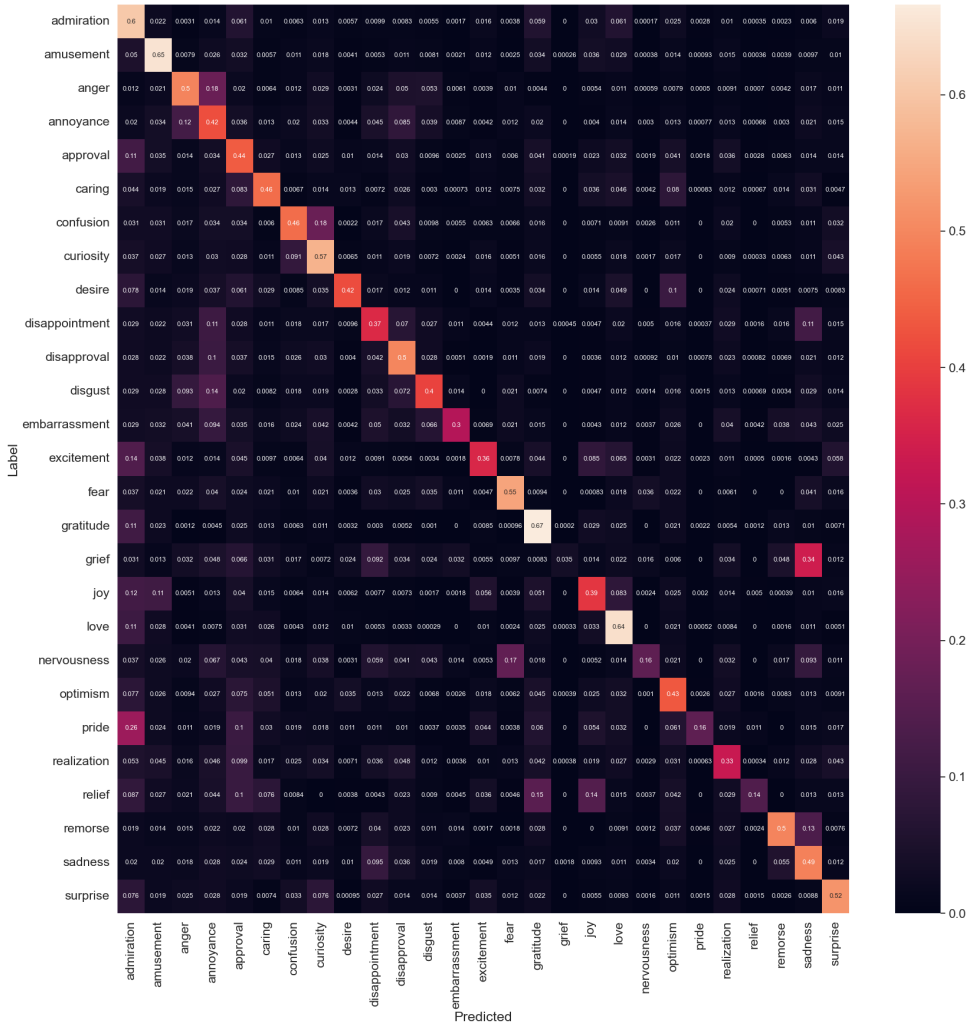
Nevertheless these results are quite good for such a nuanced task, and caught our attention when preparing our project.
We will use them as a reference for our models.

Unfortunately, the exact model that was used in this paper, has not been made available. Instead, Google released a different model called "PRADO" to accompany this dataset(which we have tested in the next section).

Luckily, several people have recreated the BERT model based on this dataset, and we were able to find a pre-trained model[22], which we will use to classify our Lexica dataset.

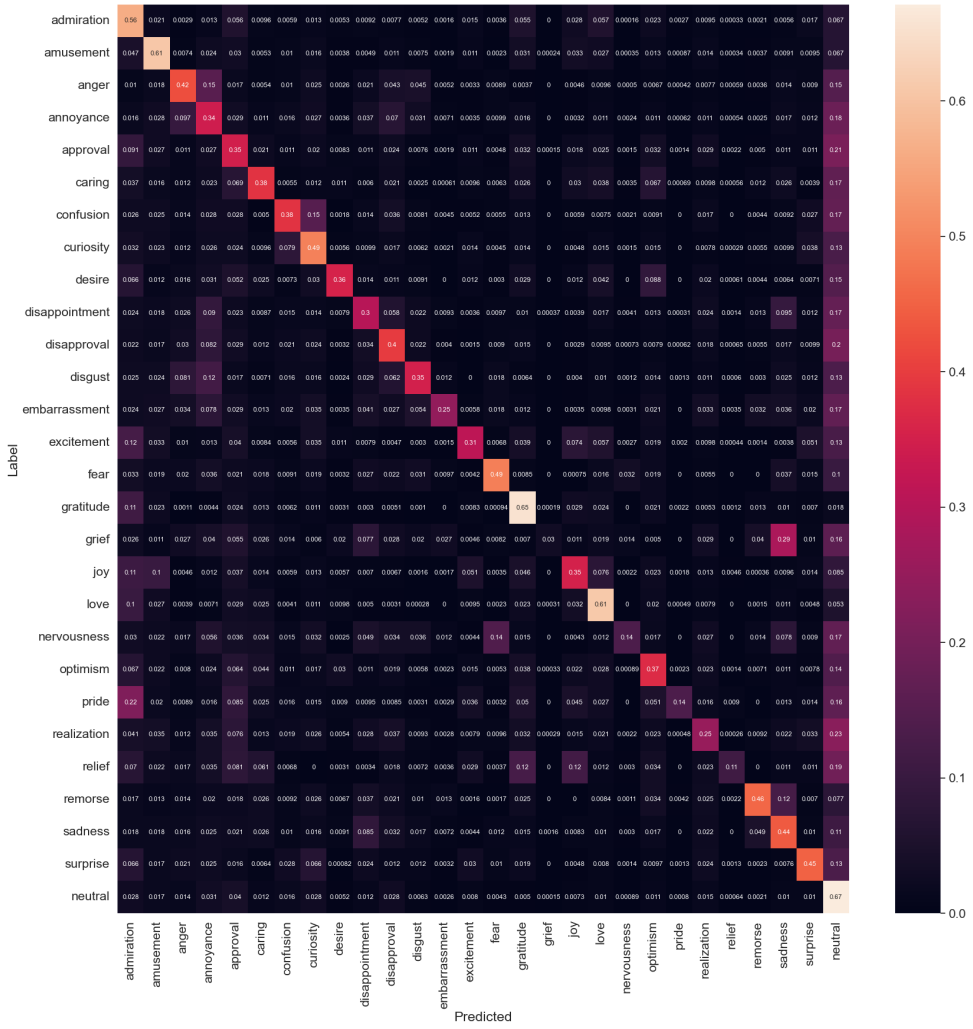
According to the author, this model achieves very close metric scores to the ones shown in the paper. And when we generated the confusion matrix, using 20% of the dataset as our validation set, we can see similar results:

Figure 5: GoEmotions BERT confusion matrix on reconstructed model.



Absent in the original graph provided in the paper, is the inclusion of the neutral label.
So when we generated the corresponding confusion matrix including the neutral label, it confirmed our hypothesis that the model is a bit skewed towards the neutral label.
That is likely the consequence of the unbalance in the dataset.

Figure 6: GoEmotions BERT confusion matrix on reconstructed model, including the neutral label.

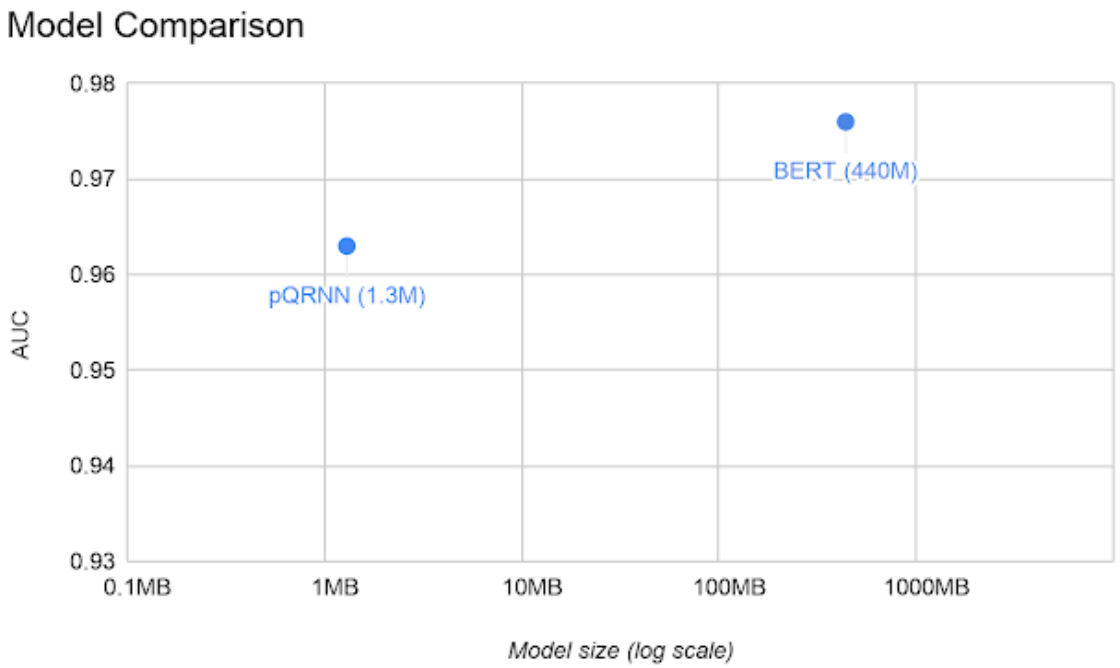


5.4 PRADO reference

As mentioned in the previous section, the GoEmotions team accompanied the paper with a different model than the one discussed in the paper, called PRADO which stands for "Projection Attention Networks for Document Classification On-Device".[19]

As the name suggests, PRADO is a very small and efficient model also developed by Google, where they claim performance that comes close to BERT.

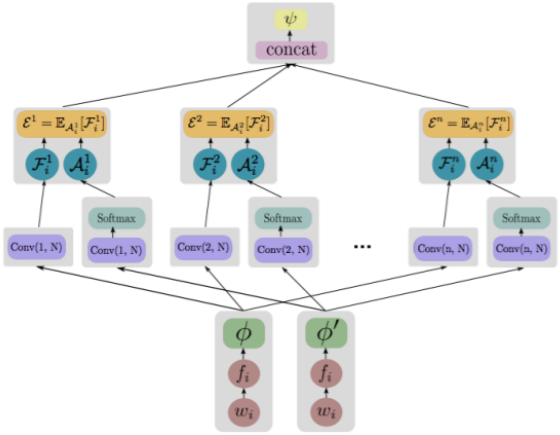
Figure 7: BERT vs pQRNN(extension of PRADO) model comparison [19]



This really caught our attention, as this makes the model much easier to train on GoEmotions.

According to Google [19], PRADO achieves this by using trainable projections with attention and convolutions, instead of transformers and a static pre-trained tokenizer, like we used in our models. In addition, PRADO is designed to learn clusters of text segments from words rather than word pieces or characters.

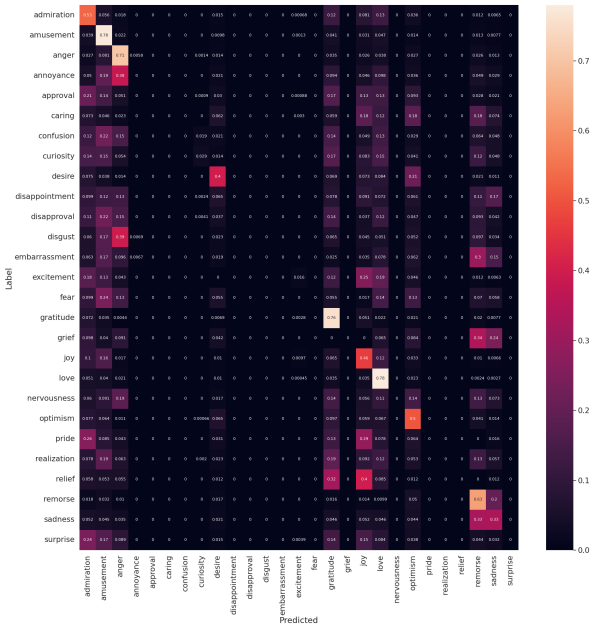
Figure 8: PRADO model architecture [19]



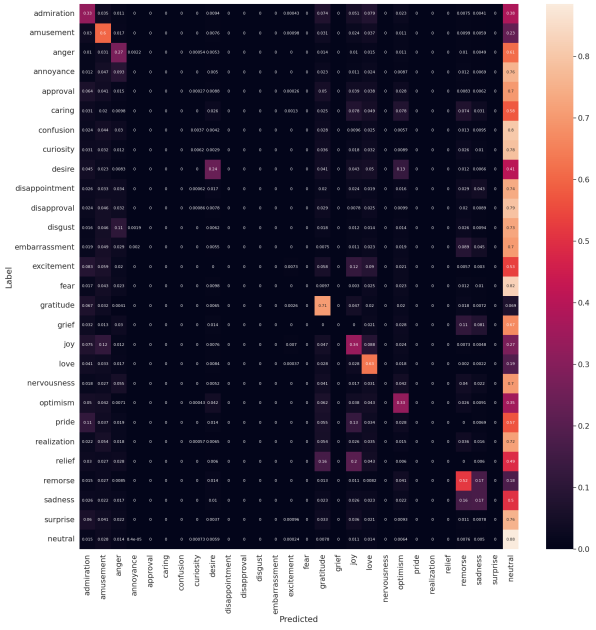
After training PRADO on the GoEmotions dataset, using the provided instructions, we got the following macro-average metrics:

Precision	Recall	F1
0.63	0.36	0.46

At a first glance, these metrics seem to confirm BERT-like performance, but during further testing we noticed that the model only understands a couple of emotions well, while others barely or not at all. This is confirmed when we generate the corresponding confusion matrices:



(a) Confusion matrix for PRADO



(b) Confusion matrix for PRADO, including neutral label

We can see that the model is very skewed towards neutral label, reflecting the unbalance in our dataset. And also we can see that the model has not notion of certain emotions such as disgust, disapproval, fear, and so on... It also seems to often confuse the few emotions that it does guess.
We suspect this is a result of the model's small size and the limitations of the dataset.

For these reasons we chose not to use this model in our project, but we are nevertheless intrigued by it and wish to maybe explore this model and it's extension in future projects as it seems very promising with the right application.

5.5 Our models

Our models all follow a similar pipeline.
We must first pass our text input, which consists of word tokens, into the tokenizer which transforms it into a list of vector embeddings.
Then it is passed to our CNN model, which will output for each of the 28 possible labels, a score between 0 and 1.

5.5.1 Tokenizer

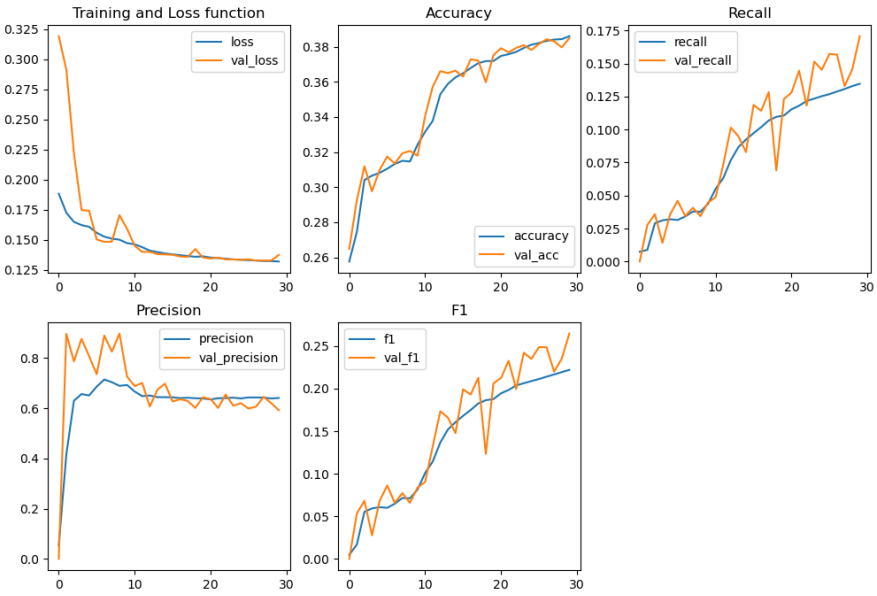
For the embeddings we used the Word2Vec[17] model, with a vector size of 300 and based on the GoEmotions corpus.

For this we first had to compile and pre-process all the sentences in our dataset.[6]
This included splitting our sentences into lower-case tokens, removing punctuation's and non-alphabetic tokens, and filtering out stop words.
This is important so that we only keep the tokens that add to the meaning of the sentence and are not there just for syntax.

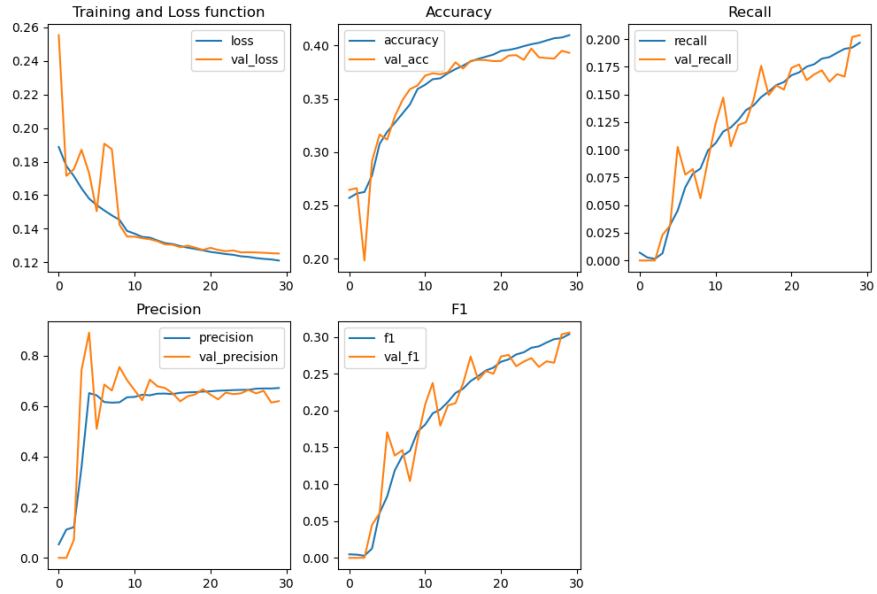
Finally a sentence must be padded a fixed length, such that the dimensions the input to our model is fixed.

Additionally we also acquired Stanford's "GloVe"[25] vectors which were pre-trained on the Wikipedia 2014 & Gigaword 5 datasets.
As they were trained on a much larger datasets, we expect the quality of these vectors to be much better than what we generated and give our models a better chance at this task.

To test this, we trained the same model on the GoEmotions dataset, using both our own embeddings and GloVe:



(a) Training results of our own word2vec embeddings



(b) Training results of GloVe embeddings

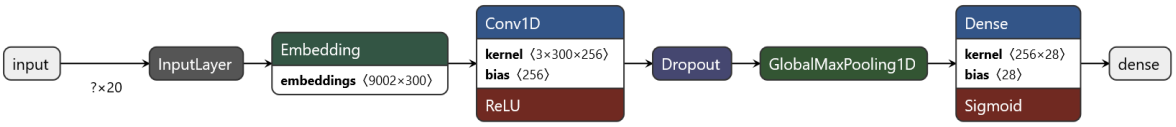
Figure 10: VGG-16 model using our own word2vec embeddings vs GloVe embeddings, the x-axis corresponds to the amount of epoch's trained, and the blue and orange lines represent the metric calculated over the training and validation set, respectively.

As we can see, using Glove embeddings we get slightly better results for the F1 and Recall metrics, while getting similar results for the rest.
This suggests that using GloVe embeddings, does give our model a small edge over using our own embeddings, but that is not the factor that is holding our models back.

Moving forward we decided to use GloVe vectors for our pipeline.

5.5.2 Naive Model

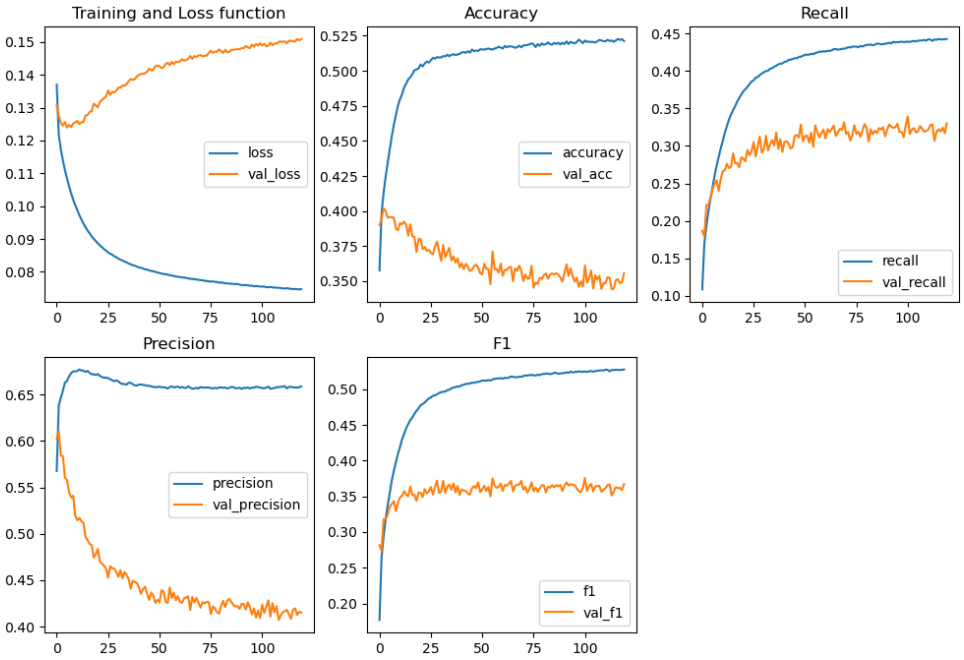
Figure 11: Naive model architecture



The architecture for our naive model is shown above, it is the bare-minimum required for a CNN classifier. A single convolutional layer for feature extraction and a dense layer for classifying these feautures.
Needless to say we did not expect it to do good, below are the training results:

Precision	Recall	F1
0.42	0.33	0.36

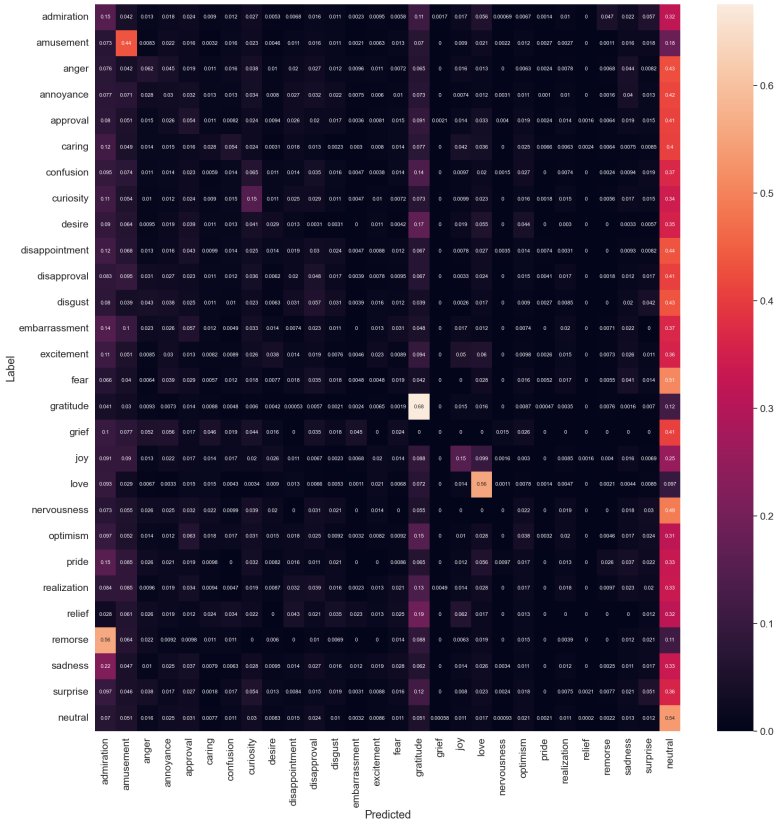
Figure 12: Naive model training results, the x-axis corresponds to the amount of epoch's trained, and the blue and orange lines represent the metric calculated over the training and validation set, respectively.



We can clearly see that after 10 or so epochs, our model started to over-fit on the training set, causing the validation results to plummet.

We can as before, generate the confusion matrix for this model:

Figure 13: Confusion matrix for naive model

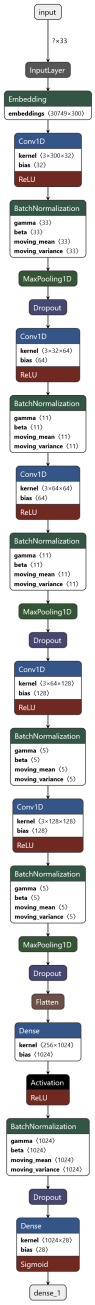


We can also see a result of the over-fitting, which is that the model will very often guess 'neutral' as that is often a safe bet. It also can't really differentiate a single emotion reliably.

When we increase the complexity of our model, we hope that it at least does better than this model.

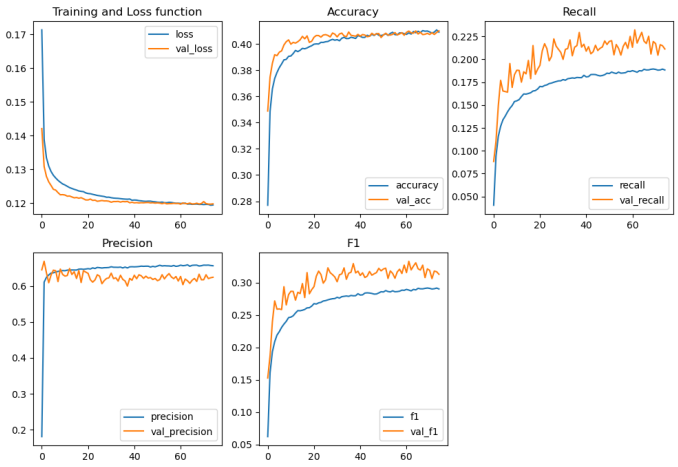
5.5.3 SmallerVGG

Figure 14: SmallerVGG model architecture

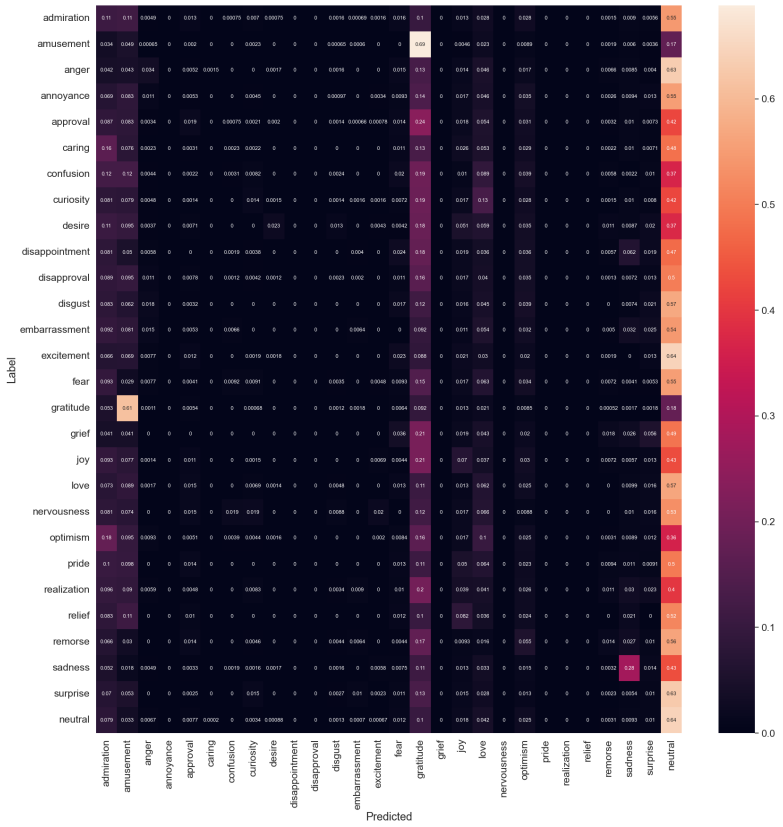


As the name suggests, this model is based on a smaller version[27] of the well known VGGNet[5] model. Below are the training results and confusion matrix:

Precision	Recall	F1
0.62	0.21	0.29



(a) Training of SmallerVGG model on GoEmotions



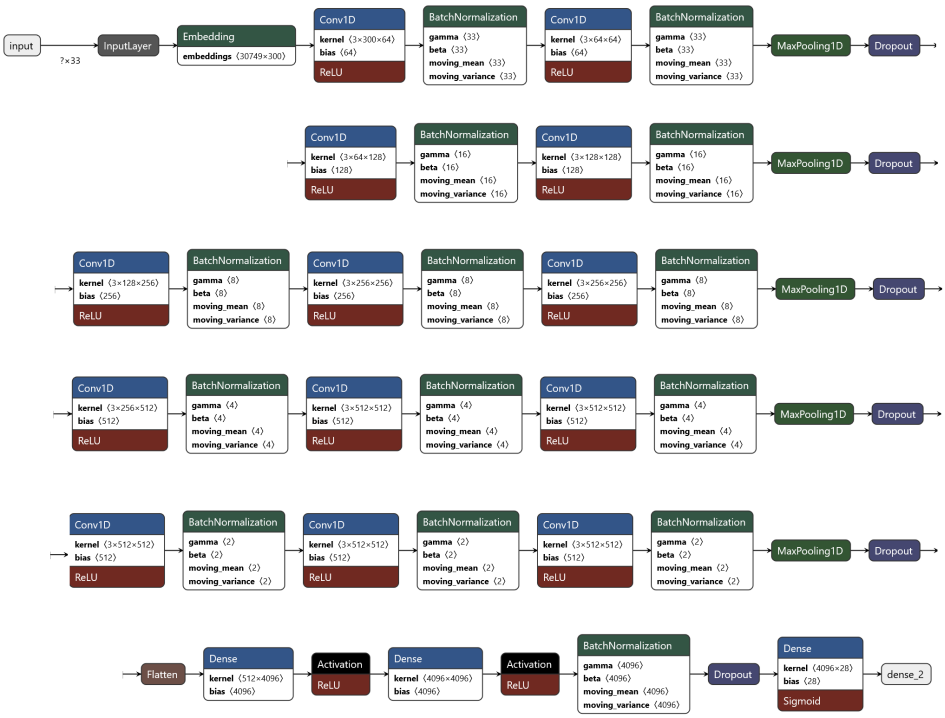
(b) Confusion matrix of SmallerVGG model

Compared to the naive model, we see a slight improvement in precision (at the cost of recall), and less signs of over-fitting, but no significant improvement in F1 scores.

When looking at the confusion matrix we also see that it's not much better than our naive model, suggesting that simply adding more convolutional layers will not significantly improve our results in this task.
Finally we tried the full VGG-16 architecture, to confirm our hypothesis.

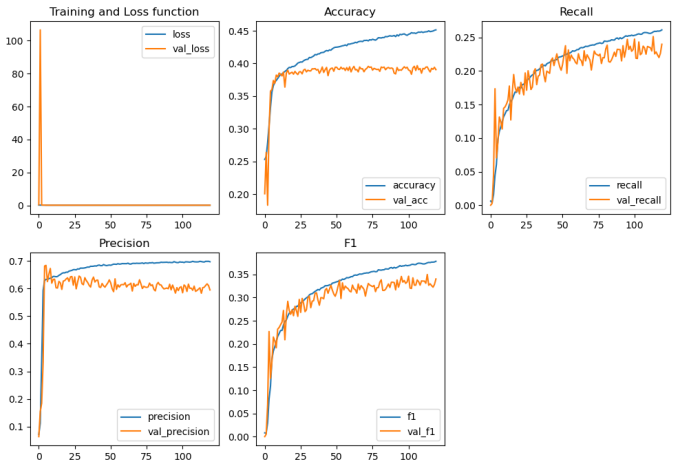
5.5.4 VGG-16

Figure 16: VGG-16 inspired architecture

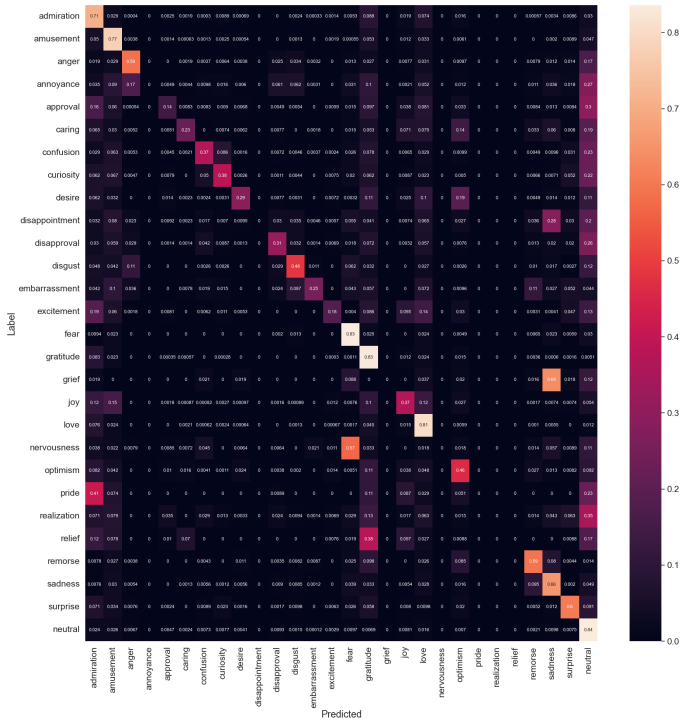


To solidify our belief that the model size does not help improve our results in this application, we created a model[29][4] closely inspired by VGG-16 and trained it on the GoEmotions dataset:

Precision	Recall	F1
0.70	0.24	0.34



(a) Training of VGG-16-like model on GoEmotions



(b) Confusion matrix of VGG-16 model on GoEmotions

While not improving significantly on the metrics in comparison to SmallerVGG, the confusion matrix looks quite different.
We can clearly see a line forming along the main diagonal, though still with a lot of confusion.

This suggests that this bigger model can have some notion of more emotion labels, though it still struggles to differentiate between them.
Given that it's such a big model, it is possible that we didn't train it for long enough due to hardware limitations. But when looking at the loss and F1 graph, we are inclined to believe that we are reaching the point of diminishing returns.

This topic could be explored further for a more certain conclusion, but for the purposes of our project, we will use the BERT model to classify our AI-prompts.
We chose to do this ahead of time instead of on the fly, as this will allow us to index the score for each emotion and allow the user to filter by threshold values.

6 Retrieval Evaluation

We were planning to use Lexica's API to evaluate our system by searching for queries and using the size of the returned array as the total number of relevant documents R. However, we have discovered that the API only returns 50 results. In addition, because the dump is outdated, We are eliminating any prompts that are not found in the dump from the returned array. This leaves us with an R value of maximum 50.

On top of that, the API's results are not solely determined by the query that was provided. It appears that the Lexica uses CLIP[26], a machine learning model developed by OpenAI, to generate text based on an image and then stores this generated text along with the image as the official prompt, rather than associating the image with the original user-provided prompt. This means that even if the original prompts are copy-pasted as queries in the Lexica search engine, the associated image could possibly not appear in the top 50 results.

To compare our search engine with Lexica's: Our search engine uses a query to search through user-provided prompts and returns an array of size k containing the top-k search results. On the other hand, Lexica's search engine uses a query to search through prompts generated by CLIP, finds the top 50 matching images, and then returns the original user-provided prompts associated with those images.

As an example, take a prompt like "blue sky" which the user provides to the art generator. The art generator returns an image with a blue sky and maybe some clouds, and both the prompt and image are fed to Lexica. But Lexica does not index that prompt, instead it will feed said image to CLIP and it will return a description of what is seen in the image, e.g. "blue sky with sparse clouds". This description is then indexed and used in the search engine.

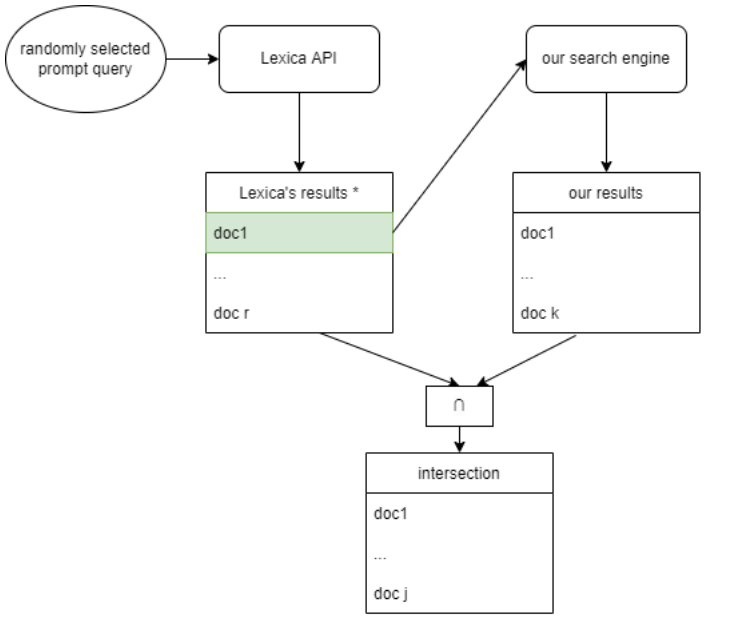
As we can see, this description may be more descriptive than the original prompt, thus allowing Lexica to find more relevant documents than if it had done a text search for the original prompt.
So we can conclude that Lexica's search engine will have a higher recall compared to our system.
This results in a tiny intersection between Lexica's results and our results.

We attempted various strategies to obtain more than 50 relevant results without success. We also tried some other creative ways to evaluate our system. One approach we tried for calculating some kind of precision@k is described in a visual way in figure 18. We calculate

$$(precision@k)' = \frac{|intersection|}{k}$$

which is a lower bound of precision@k because $|intersection|$ can only increase, as our search results are limited to 50 or fewer. The average result we get from running 661 queries is 18.5% which, again is a lower bound of the real precision@k value.

Figure 18: Visual model of our evaluation system



7 Conclusions

Throughout this project we have learned a lot about creating and deploying information retrieval systems in practice, the challenges of indexing and classifying large datasets, and about different NLP machine learning models for multi-label text classification.

Keeping in mind the limitations of our models and datasets, this project gave us a look at how data retrieval systems will play an important role in the ever-increasing pool of machine generated art, and how machine learning models can enhance such systems.

8 GitHub Repository

The GitHub repository can be accessed via this link.

References

[1] *30% of Google's Emotions Dataset is Mislabeled*. Dec. 2022. URL: https://www.surgehq.ai/blog/30-percent-of-googles-reddit-emotions-dataset-is-mislabeled.

[2] Anton_Shuvaev. "Elasticsearch". In: *JetBrains Marketplace* (Nov. 2022). URL: https://plugins.jetbrains.com/plugin/14512-elasticsearch.

[3] *Apache Lucene*. Nov. 2022. URL: https://lucene.apache.org.

[4] blurredmachine. "VGGNet-16 Architecture: A Complete Guide". In: *Kaggle* (July 2020). URL: https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide.

[5] Gaudenz Boesch. "VGG Very Deep Convolutional Networks (VGGNet) - What you need to know". In: *Viso* (Dec. 2021). URL: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks.

[6] Jason Brownlee. "How to Clean Text for Machine Learning with Python - MachineLearningMastery.com". In: *MachineLearningMastery* (Aug. 2019). URL: https://machinelearningmastery.com/clean-text-machine-learning-python.

[7] Contributors to Wikimedia projects. *Artificial intelligence art - Wikipedia*. Dec. 2022. URL: https://en.wikipedia.org/w/index.php?title=Artificial_intelligence_art&oldid=1127712343.

[8] Contributors to Wikimedia projects. *BERT (language model) - Wikipedia*. Dec. 2022. URL: https://en.wikipedia.org/w/index.php?title=BERT_(language_model)&oldid=1127646245.

[9] Contributors to Wikimedia projects. *Convolutional neural network - Wikipedia*. [Online; accessed 17. Dec. 2022]. Nov. 2022. URL: https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=1123805333.

[10] Contributors to Wikimedia projects. *DALL-E - Wikipedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=DALL-E&oldid=1120092751.

[11] Contributors to Wikimedia projects. *Midjourney - Wikipedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Midjourney&oldid=1121592716.

[12] Contributors to Wikimedia projects. *Stable Diffusion - Wikipedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Stable_Diffusion&oldid=1121441323.

[13] Dorottya Demszky et al. *GoEmotions: A Dataset of Fine-Grained Emotions*. 2020. URL: https://ai.googleblog.com/2021/10/goemotions-dataset-for-fine-grained.html.

[14] Dorottya Demszky et al. *GoEmotions: A Dataset of Fine-Grained Emotions*. 2020. DOI: 10.48550/ARXIV.2005.00547. URL: https://arxiv.org/abs/2005.00547.

[15] Elastic. *Elasticsearch*. Dec. 2022. URL: https://github.com/elastic/elasticsearch.

[16] *Elasticsearch DSL*. July 2022. URL: https://elasticsearch-dsl.readthedocs.io/en/latest/#.

[17] *Gensim: Word2vec embeddings*. May 2022. URL: https://radimrehurek.com/gensim/models/word2vec.html.

[18] Jeremy-Fuller. *Prompts*. 2022. URL: https://github.com/Jeremy-Fuller/Prompts.

[19] Prabhu Kaliamoorthi, Sujith Ravi, and Zornitsa Kozareva. "PRADO: Projection Attention Networks for Document Classification On-Device". In: *ACL Anthology* (Nov. 2019), pp. 5012–5021. DOI: 10.18653/v1/D19-1506.

[20] *Lexica*. 2022. URL: https://lexica.art.

[21] Ml2021dsb. "Multi Class Text Classification using CNN and word2vec". In: *Medium* (Dec. 2021). URL: https://ml2021.medium.com/multi-class-text-classification-using-cnn-and-word2vec-b17daff45260.

[22] monologg. *GoEmotions-pytorch*. Dec. 2022. URL: https://github.com/monologg/GoEmotions-pytorch.

[23] *NLP Essential Guide: Convolutional Neural Network for Sentence Classification*. June 2021. URL: https://cnvrg.io/cnn-sentence-classification.

[24] *pandas - Python Data Analysis Library*. Dec. 2022. URL: https://pandas.pydata.org.

[25] Jeffrey Pennington. *GloVe: Global Vectors for Word Representation*. June 2021. URL: https://nlp.stanford.edu/projects/glove.

[26] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. DOI: 10.48550/ARXIV.2103.00020. URL: https://arxiv.org/abs/2103.00020.

[27] Adrian Rosebrock. "Keras and Convolutional Neural Networks (CNNs)". In: *PyImageSearch* (Feb. 2021). URL: https://pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns.

[28] sarnthil. *unify-emotion-datasets*. Dec. 2022. URL: https://github.com/sarnthil/unify-emotion-datasets/tree/master/datasets.

[29] Rohit Thakur. "Step by step VGG16 implementation in Keras for beginners". In: *Medium* (Dec. 2021). URL: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c.

[30] *The Go Programming Language*. Dec. 2022. URL: https://go.dev.