Study of classification and summarization techniques applied to the BigPatent dataset

Marco Braga, Andrea Maver

January 12, 2022

Keywords: patent, text, classification, summarization

1 Introduction

In the last five years, more than one hundred and fifty thousand patents have been published in Europe. This implies that this large quantity of texts must be analyzed and classified to be organized in enormous dataframes. Being able to automatically classify a long text, like the one of a patent, can be a very useful resource to simplify human work and help with the storage of the documents. Also, the possibility of having a computer generated summary of a text can help to have a general idea of its content, without having to read it all.

The aim of this project is to study different algorithms and techniques of text classification and text summarization, applied to a dataset of American patents, and to evaluate all the performances to find the best ones. A possible application of the results of this research could be the implementation of the top models, by agencies or organizations, in order to manage the large quantity of incoming patents in an automatic and fast way.

For this type of research, the chosen data to be analyzed will be from the BigPatent dataset, a very large dataset with over a million of different documents, largely used for text mining tasks.

The paper is divided into the following sections: a more in depth explanation of the dataset and how it is constituted; a brief theoretical explanation of the methods used to analyse the dataset; the actual part of analysis, consisting of both classification and summarization, and the different methods that have been tried; a discussion about the results and the final conclusions and possible further developments.

2 Data

The BigPatent dataset consists of 1.3 million records of U.S. patent documents along with human written summaries. Each US patent application is filed under a Cooperative Patent Classification (CPC) code. There are nine such classification categories:

- A: Human necessities
- B: Performing operations; Transporting
- C: Chemistry; Metallurgy
- D: Textiles; Paper
- E: Fixed constructions
- F: Mechanical engineering; Lightning; Heating; Weapons; Blasting
- G: Physics
- H: Electricity
- Y: General tagging of new or cross-sectional technology

The dataset has two columns: 'description', the detailed description of the patent, and 'abstract', a human-made summary of the document's contents. Data can be found at [1].

The data is organized in a .tar file divided into *train.tar.gz*, *test.tar.gz*, *val.tar.gz*. Because of the high computational costs of the

tasks, only the *train.tar.gz* set has been chosen. After unzipping, the data is divided into nine folders, one for each of the classification categories described above. In every folder there are many file .gz, and each of them contains a different number of documents. The first four of this files are selected to create the dataset that, after a shuffle of its rows to eliminate consistencies, is finally composed by 45027 rows and 3 columns: description, abstract and topic.

For this project, only the categories from A to H have been taken into consideration, because the last one (Y), consisting of 'General tagging of new or cross-sectional technology', created a lot of difficulties during the classification task. These problems are due to the fact that this category comprehends a lot of different types of patents, and it's a sort of generic repository for the ones that aren't about a specific subject.

3 Methodological aspects

This part of the paper explains some of the theory behind the different tasks performed and the pre-processing steps necessary to facilitate the understanding of the text by computers.

3.1 Tokenization, stemming and lemmatization

The first step for text mining applications is pre-processing, which will be described in detail later. After removing some noise, like URLs and very common words, tokenization and stemming/lemmatization need to be applied to the documents; different settings are tried in order to find the best working ones for the classification and summarization tasks.

The first approach that has been used is tokenization based on white space, in which each token is a sequence of characters that doesn't contain any white space; basically, the system creates a list in which every word is a token. After this, the Porter stemming is applied, which reduces inflected or derived words to their root form by following a sequence of set rules.

The second approach requires the *SpaCy* library to tokenize words and apply lemmatization based on Part of Speech (POS). The algorithm firstly divides the document in tokens of words, then assigns to each one of them its corresponding part of speech, which could be article, pronoun, verb, etc. After POS is applied, the lemmatizer starts working, bringing every word to its root form; this process will transform words like 'are' and 'being' both to 'be', the base form of the verb 'to be'.

3.2 Document representation

After these pre-processing steps, the documents must be represented; the chosen representations are a Tf-Idf matrix and Doc2Vec.

Tf-Idf is a technique to weight the importance of the words in a text based on two factors: the Term Frequency of each term in each document and the Inverse Document Frequency. The basic idea is that a word's relevance increases if it appears very frequently in a document, but decreases if it also appears a lot in the whole collection. The collection is therefore represented as a table in which the documents are in the rows and the tokens in the columns; the value in every cell is the Tf-Idf weight of that token in that document.

Doc2Vec is an architecture composed of two algorithms, just like Word2Vec: one called Paragraph Vector - Distributed Bag of Words (PV-DBOW), similar to the Skip-Gram model in Word2Vec, except that a new vector representing paragraphs' IDs is added. After choosing a target word, a neural network is trained to predict the vector of its surrounding terms in the given paragraph. The second algorithm is Paragraph Vector - Distributed Memory (PV-DM), similar to CBOW in Word2Vec, which, on the contrary, predicts target words based on their context.

3.3 Text classification

Text classification is a supervised machine learning technique whose aim is to predict to which class a new text will belong; in this

case, it's a Single Label Multi-Class classification, in which only one label can be assigned to the document. The classes are the eight previously described (Human Necessities, Transporting, Chemistry, Textiles, Fixed Constructions, Mechanical Engineering, Physics, Electricity).

To perform this classification, different techniques have been tried: Decision Tree, Random Forest Classifier, Linear SVM and K-Nearest Neighbour, which will be briefly explained.

- Decision Tree: the simplest method; separates the data belonging to different classes, deriving a tree representation that is built through a learning phase, a recursive heuristic procedure based on the 'divide and conquer' scheme.
- Random Forest Classifier: a meta estimator that fits a number of decision trees on various sub-samples of the dataset and uses averaging to improve accuracy.
- Linear SVM: a constrained optimization model whose aim is to find the hyperplane (linear function) that maximizes the minimum distance between the hyperplane and data.
- K-Nearest Neighbour: an object is classified studying its neighbourhood, i.e. the K-closest training observations; the assigned class is the most common label among its neighbours.

Since Linear SVM has high computational costs, it's useful to reduce the dimension of the training data through Singular Value Decomposition, a numerical algorithm based on linear algebra. Any matrix $M \in Mat_{nxm}(\mathbf{R})$ can be decomposed as the product of three matrices:

- U: a nxl matrix where the n rows correspond to rows of the original matrix M, and the l columns represent new features in a new latent space.
- S: a diagonal lxl matrix of singular values, expressing the importance of each dimension feature.
- V^T: a transposed lxm matrix where the m columns correspond to the columns of the original matrix and the l rows correspond to singular values.

Between the three, only matrix U is considered to proceed with the analysis, so the data will be studied in a smaller l-dimensional space, where l is defined a priori.

In order to evaluate the performances of the various classification models, and for binary classification problems, a confusion matrix can be a useful tool (table 1).

	Real True	Real False
Predicted True	TP	FP
Predicted False	FN	TN

Table 1: Confusion Matrix Λ

Using this matrix, a measure expressing the effectiveness of model, called accuracy, can be calculated:

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

Vanilla accuracy is an adaptation of standard accuracy created to fit a multi-class classification problem; let C be the set of all the classes c_i considered for the problem, vanilla accuracy is defined as:

$$VA = \frac{\sum_{c_i \in C} \Lambda_{ii}}{\sum_{c_i \in C} \sum_{c_j \in C} \Lambda_{ij}}$$

In order to obtain a more robust estimate of the classifiers' goodness, always considering vanilla accuracy, cross validation has been performed. The dataset is partitioned into K disjoint subsets D_k, where $k = 1, ..., K$. The learning-testing operations are then performed this way: at every k-th iteration, the training set is composed of all the other subsets but the k-th one, which is used as validation set. The operation ends when every subset has been tried as test set; basically the learining phase is repeated k times, always with different validation sets. The final vanilla accuracy is calculated by averaging the accuracy of the single iterations.

3.4 Text summarization

Text summarization is a natural language process task aimed at creating an automatic abstract of a text. There are two main types of summarization: extractive and abstractive. An extractive summary is created using the same sentences present in the original text. A score is given to every phrase based on its importance; there are different approaches for assigning these scores, like the ones based on topical relevance (a phrase is important if it contains words regarding the topic of the whole text) and others based on some set of features. In this project, a graph-based approach has been chosen to create the summaries.

An abstractive summary, on the other hand, is created using completely new words and sentences, like the one a human would write. Because of that, automatically producing an abstract in this way is very computationally consuming. Usually, using a neural network is fundamental to complete this task successfully; with a bunch of data, and a lot time, a dedicated model could be trained and tested, thanks to the abstracts provided in the dataset. However, on the web there are plenty of pre-trained models, specific for natural language processing tasks; *Pegasus*, from Google, is one of them and it has also been pre-trained on the Big-Patent dataset. *Pegasus* is a recently developed encoder-decoder model, which provides similar results to other competitors while being very slim and therefore much less computationally expensive.

The last part of the analysis concerns the evaluation of the results of the two summarization algorithms. Evaluating an abstract is quite a difficult task, but thanks to the ground truth present in the dataset, the *Rouge* metric can be calculated. The idea behind this measure consist in counting the number of common n-grams (usually 1-gram or 2-gram) between the known abstracts and the newly created ones; there are also some variations that take into account the longest common sequence or that allow other words in-between the n-grams. Even tough the best way of evaluating abstracts remains human assessment, *Rouge* values can provide a general idea of the quality of the summarization.

The *Rouge-n* metric is defined as follows: let p be the number of common $n-$grams between the reference abstract and the obtained summary and q be the number of n-grams from the reference summary;

$$Rouge\text{-}n = \frac{p}{q}.$$

Rouge-l introduces the concept of longest common subsequence (LCS) among the two sequences of text. The idea is that the longer the LCS between two summary sentences, the more similar they are.

4 Data analysis

This section will present all the preliminary procedures of pre-processing of the text needed to make the documents usable by computers and will go in depth about the techniques and implementations of the classification and summarization algorithms.

4.1 Pre-processing

The first step in the pre-processing phase is to check if the dataset contains any null values on the different columns; thankfully it doesn't, so no other action regarding missing values is needed. Then, the actual cleaning and removal of unwanted parts can start: all the texts are brought to lower case and possible URLs present in the patents are eliminated, also like punctuation, numbers, white spaces and stop-words. It's important to note that dots and commas were not removed form the texts, because they'll be essential in the summarization part. In addition, a new copy of the dataset is created, just after eliminating URLs and punctuation. It will be needed for the summarization task and will be subjected to a slightly different pre-processing, but will be better addressed later.

Stop-words are terms which are generally filtered out before a natural language processing task, being too common and with very little significance, and would end up compromising the quality of the analysis.

These words are usually articles, prepositions, pronouns and conjunctions, but the list can be expanded by adding specific terms of the analysed dataset. In particular, after a search for the most common words, this are the patent specific terms added to the stop-words, that will therefore be eliminated from the patents' texts: 'position', 'step', 'embodiment', 'first', 'product', 'invention', 'process', 'show', 'example', 'within', 'form', 'also'.

Then, both the Porter stemming algorithm and the *SpaCy* lemmatizer are applied to the documents, creating two new columns made of lists of clean tokens, very similar to each other but that can potentially bring to different results later. Also, from the remaining tokens, the ones with less than three characters are removed, to make sure to eliminate strange abbreviations and leftovers from the previous steps. A simple function to find the new most common lemmatized tokens reveals that 'surface', 'device', 'portion', 'material', 'provide' and 'include' are now some of the most recurring terms.

Coming back to the summarization specific dataset, it was left with only the original description, the pre-processed text, abstract and topic. First of all, only 2500 patents belonging to the B class (Performing Operations; Transporting) are selected. This reduction is needed because of the computational limitations in running the summarization algorithms; having more computational power, this number can be easily increased.

The first step is to tokenize the sentences from the pre-processed documents: this passage is fundamental, because when the extractive summary will be created, it will need to use these clean sentences, otherwise it would make very little sense. Then, a further cleaning is run on these sentences, removing numbers, white spaces and stop-words as before; the resulting phrases are stored into a new column. The final summarization-specific dataset is composed of 5 columns: abstract, topic, pre-processed document, clean sentences and normalized sentences. Both the clean and the normalized sentences columns are made of lists of phrases, which will be needed for the following summarization steps.

4.2 Text classification

4.2.1 Tf-Idf based

The dataset is divided into two with an 80:20 proportion: the larger part represents the training set, while the other is the test set, which will serve as never before seen data. The training set is further divided into two subsets with the same proportions: 80 percent for the actual training and 20 percent for the validation. The k value for cross validation is set to three, so the learning phase will be repeated three times on different subsets of training and validation. For each iteration of cross-validation, a Tf-Idf vectorizer is instantiated on the training data, and then run on the validation set to test its correctness and obtain the required representation of the documents.

A set of hyperparameters must be fixed: the minimum and maximum threshold of the terms' document frequency, indicating if a word has to be considered or eliminated while creating the dictionary (an integer indicates the number of documents); the maximum number of features and the n-gram range, which indicates the types of n-grams to be taken into account. The selected values are the following: $mindf = 2$, $maxdf = 0.5$, $maxfeatures = [1000, 5000, 15000]$, $ngramrange = [(1,1),(1,2),(2,2)]$. This setup ensures that different solutions are tested, to find the best ones. Then, all the different models can be run, saving their vanilla accuracy into a dictionary; to test the Linear SVM correctly, the dimensions of the dataset are reduced through the application of Singular Value Decomposition. These processes are repeated both for the the tokens obtained with Porter stemming and the ones obtained through lemmatization with part of speech, to be able to compare their different performances.

4.2.2 Doc2Vec based

Because of its high computational costs, Doc2Vec will be only applied on the texts pre-processed with lemmatization. The first

step is to decide the vector's size, which in this case will be three hundred and fifty, and the number of epochs to train the model, which will be thirty; a higher number of both of this parameters could probably provide better results, but it would also be very time consuming and expensive. After defining the vocabulary from the words in the texts, a model is trained on the original train set; then, the newly trained model is applied on both the train and test sets. Once the representation is obtained, a Random Forest Classifier is applied to get some classification results.

4.3 Text summarization

4.3.1 Estractive summarization

First of all, a Tf-Idf vectorizer is defined; it is then used to create a similarity matrix between the normalized sentences. Employing this matrix, a graph is created, in which the nodes correspond to the single phrases, and the edges to the similarity scores between them. To assign an importance score to the sentences, *PageRank* is run on the graph. *PageRank* is an algorithm used to rank textual documents in the web and its key idea is that a document, or in this case, a phrase, is important if it has a lot of incoming links, or if a meaningful one points directly at it. Using this concept, every sentence receives a relevance score. The remaining part is just a matching between the indices of the best four normalized sentences and the ones of their clean counterpart, to create the extractive summary. The number four was chosen after looking at the average number of phrases constituting the ground truth; it's a very basic method, but allows to avoid making a random choice based on no asset. It's important to notice that these passages have to be repeated for every singular document, so 2500 matrices, graphs and *PageRank* runs; this can make the task very computationally expensive, but again, with better machines the number can be raised.

4.3.2 Abstractive summarization

For this type of summarization, the model doesn't require the division of the texts into sentences, the pre-processed document is enough; as a reminder, these are the original patents without URLs and punctuation. The first step is to download both the BigPatent pre-trained *Pegasus* model and its tokenizer; this provides better results than using a generic natural language processes model. Then, the encoder-decoder part of the model is defined, with some parameters concerning the truncation of the tokens and their padding. The algorithm is applied just to a small part of the dataset, consisting of 200 patents; this number enables to run a significant demonstration of how *Pegasus* works and its results, without being too time consuming.

4.3.3 Rouge evaluation

The *Rouge* part covers the evaluation of the two summarization methods. The metric requires just the reference summary and the new ones, and outputs the scores of three of its different variants: Rouge-1, Rouge-2 and Rouge-l; for each one of them, the algorithm calculates precision, recall and f-measure. After the values have been put into new columns, an average score is calculated, to have a general idea of the quality of the models and to make an easy comparison of the two.

5 Results

This section will provide an in-depth explanation of the results of the two tasks.

5.1 Classification

The vanilla accuracy of every model is represented in the following tables, the first concerning the data with Porter stemming (2) and the second related to the lemmatized data (3), only considering 15000 words and 1 or 1 and 2-grams. In table 5 in the Appendix is possible to check the results for all the possible combinations of parameters.

Applying Porter stemming, Random Forest is the best model, achieving a vanilla accuracy of 63.5 percent with 1 and 2-grams; K-Nearest Neighbour is very close, with 62.6 percent, always with 1 and 2-grams. Both

	1-gram	1 and 2-gram
Decision Tree	0.488	0.488
Random Forest	0.632	0.635
Linear SVM	0.262	0.328
KNN	0.615	0.626

Table 2: Vanilla accuracy of the models with Porter Stemming

Decision Tree and Linear SVM don't even reach the 50 percent mark, which is not great. Using different settings of n-grams basically doesn't produce any difference in the results.

	1-gram	1 and 2-gram
Decision Tree	0.497	0.494
Random Forest	0.633	0.637
Linear SVM	0.313	0.302
KNN	0.615	0.627

Table 3: Vanilla accuracy of the models with lemmatization

Lemmatization results are very similar to the stemming ones, with the best performing model still being Random Forest, which associates correctly 63.7 percent of the patents, followed by K-Nearest Neighbour, with almost identical performances, then Decision Tree and finally Linear SVM. Like before, the usage of 1-grams or 1 and 2-grams doesn't really change the results for the algorithms, that at most are 3 percent more accurate.

A further run of the top performing model, Random Forest, on the test set with just 1-2 n-grams, gives a final leading vanilla accuracy of 65.5 percent.

Here are some reflections for each of the four classification methods:

- Decision Tree: it gives bad results, probably due to its simplicity; in fact, Random Forest, which puts together a lot of single trees, provides an higher accuracy of approximately 15 percent.
- Random Forest: it provides the best results, which are good but not great; considering more decision trees could certainly bring better performances, but would also be more expensive.
- Linear SVM: the selected SVM has only 150 dimensions, a very low number compared to the 15000 of the original problem, so the results can't be very accurate. Another way of improving, beyond the increment of the number of dimensions, could be using a non-linear SVM with kernels, but in both cases the task becomes more computationally costly.
- K-Nearest Neighbour: it's the simplest technique and doesn't require a real model underneath, but works almost as fine as the more complex Random Forest. Maybe with a better representation from Doc2Vec it could provide the best results, given that similar documents should have similar representation.

On the other hand, the Doc2Vec representation tested with a Random Forest Classifier gave some mediocre results, reaching an accuracy of 48.5 percent. This could be due to the fact that the model wasn't trained for a lot of epochs and with small number of data for a neural approach. Maybe applying stemming could be an interesting solution and development.

5.2 Summarization

To show the results of summarization, both human judgement and *Rouge* scores would be required. Looking at some examples of the two summaries, an immediate consideration is that the abstractive summaries look much more similar to the reference abstract than the extractive one; the obvious reason is that the original is human-made.

Considering the extractive ones, they all seem a bit strange and sometimes make very little sense if looked at as a whole. That's because the text of a patent often contains practical instructions on how to build a device or how a new invention has been developed; there are also lots of references to images, which make the raw text very hard to understand. Looking at the single sentences, though, and in particular to the terms, it's possible to find some of the words that appear in the reference and that define the subject of the patent.

On the other hand, the abstractive summaries seem a lot more accurate: the model was able to find what the document is about almost every time, and created a proper abstract, describing what the main topic is and even adding some useful information to have a more complete understanding. The syntactical and semantical use of words, as their placement inside the sentences, is very good but not perfect, in fact this is quite a difficult task for a machine. To sum it up, the results are very understandable and capture the essence of the documents.

The average *Rouge* scores selected to compare the two techniques are the F-measure of Rouge-1 and Rouge-l; the results are shown in the table 4.

	Rouge-1	Rouge-l
Extractive	0.241	0.198
Abstractive	0.340	0.294

Table 4: Average Rouge values (F-measure) for the two summarization methods

Looking at the scores, it's clear that the abstractive summarization performs in a better way than the extractive one, reaching over 10 percent more words in both of the categories. The raw numbers might not suggest great overall results, but again, looking the abstracts *Pegasus* produced, it's safe to say that they're quite reasonable and meaningful. The low scores might be due to the fact that the reference and the new summary have only few common words, but that doesn't necessarily indicates the incorrectness of the automatically generated one.

A couple of randomly chosen examples of the reference abstract and the new ones are shown in the Appendix.

6 Conclusions and possible developments

The classification task gave some interesting results, but also highlighted some very clear problems. First of all, the computational costs were a huge limit, and represented the most challenging issue. The point lies in finding a good trade-off between the number of documents to be analyzed and the goodness of the final results: with few texts, the models will run quickly, but the results would probably be very bad; on the other hand, using a large sample of data provides better results, but it also becomes very expensive. Also, this particular type of textual data, the patents, had its own difficulties: the documents are usually very long, and this makes the classification harder, compared to some short ones like tweets and online reviews. In addition, a lot of specific pre-processing was required, and possibly some other more particular adjustments could have been made. A run of the various algorithms on all 1.3 million data would certainly provide better results, but it would also require very high computational resources and a lot of time.

Having said that, some of the results were not bad: Random Forest and K-Nearest Neighbour both produced an accuracy of over 60 percent. Between stemming and lemmatization, the second one performs just slightly better, but the whole process is slower, because it also needs to assign a part of speech to each word. A possible development regarding the Porter stemming could revolve around trying to create the tokens with nltk instead of white spaces: it would certainly be more time consuming, but, considering stemming's speed and the fact the the performances were almost identical to the lemmatization ones, expanding in this way could be worth. The different configurations of n-grams didn't provide significant differences in the results; anyway, using 1 and 2 n-grams might be the best way, due to its higher comprehensiveness. The Doc2Vec representation with a Random Forest classifier gave some average results; again, the possible improvement could revolve around the increase of the number of data to train it, and maybe trying some other different classification methods. This type of implementation could be the one with the highest potential for the future, but it would also require a lot of time and computational resources.

For the summarization part, it is clear that an approach providing abstractive resumes better fits the task of summarizing

patents. That's because of the nature of the documents itself: having a sequence of instructions inside the text, which is an essential part of all patents, causes a lot of problems in finding singular sentences that correctly give the general idea of the patent. Having said that, the model often chose phrases which contained relevant words, but that put together made little sense. It's possible that other approaches, not based on a graph representation, could work better, like the ones analyzing topical relevance of the sentences or maybe even some based on neural networks. *Pegasus* is clearly a better fit for this type of task, and provided some really good results; with a bit more preparation, it could easily be implemented into some programs by organizations or agencies. A possible advancement could be creating a custom model, trained just on this type of data, given that the original dataset is composed of over 1.3 million patents and provides human-made abstracts.

References

[1] Eva Sharma. *BigPatent, summarization dataset*: https://evasharma.github.io/bigpatent/

[2] Eva Sharma, Chen Li, Lu Wang (2019). *BIGPATENT: A Large-Scale Dataset for Abstractive and Coherent Summarization*

[3] Yang, Y., Chen, B., Yang, Z. (2019). *An Algorithm for Ordinal Classification Based on Pairwise Comparison*

[4] Nenkova, A., McKeown, K. (2012).*A Survey of Text Summarization Technique*, s. Springer Science+Business Media

[5] Peter J. Liu, Yao Zhao (2020). *PEGASUS: A State-of-the-Art Model for Abstractive Text Summarization*

[6] Jingqing Zhang, Yao Zhao, Mohammad Saleh, Peter J. Liu (2020). *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*

Appendix

	1-grams			(1,2)-grams			2-grams		
	1	5	15	1	5	15	1	5	15
DT	0.472	0.494	0.497	0.470	0.492	0.494	0.384	0.419	0.439
RF	0.629	0.640	0.633	0.625	0.644	0.637	0.503	0.558	0.576
SVM	0.359	0.301	0.313	0.350	0.311	0.302	0.240	0.228	0.249
KNN	0.559	0.615	0.615	0.556	0.616	0.627	0.448	0.519	0.557

Table 5: Vanilla accuracy with lemmatization and different configurations

Reference	an aspect of the present invention provides an apparatus for converting images of vehicle surroundings that includes , at least one camera configured to start , upon receiving a synchronizing signal , photographing the surroundings of a vehicle and outputting image data representative of the photographs , an output memory configured to store image data to be displayed on a display installed in the vehicle , a pattern memory configured to store destination addresses of the output memory , and an image converter configured to generate the synchronizing signal , obtain the image data from the camera , and transfer part or the whole of the image data to the output memory according to the destination addresses stored in the pattern memory .
Extractive summary	and the display 5 displays the display data 40 . the output memory 4 has a bank for receiving image data and another bank for outputting display data to the display 5 . as mentioned above the output memory 4 provides the display 5 with the stored data as display data 40
Abstractive summary	An apparatus for converting images of vehicle surroundings includes a plurality of cameras for photographing the surroundings of a vehicle, an image converter for converting images received from the plurality of cameras, and a display for displaying the images converted by the image converter.

Table 6: Summarization example 1

Reference	a self - generating gas pressure apparatus such as an expandable closed pouch for placement within a container from which a fluid therein is to be dispensed under pressure . the apparatus has a plurality of internal compartments formed by pressure - rupturable seals and containing respective chemical compounds which when mixed upon adjacent - compartment seal rupture produce a gas . at least one of two adjacently - housed chemical compounds has in addition thereto a nucleating agent such as diatomaceous earth which acts to more rapidly force gas generated in the reaction of the adjacently - housed chemical compounds out of solution and thereby provide an operative pressure to the apparatus more quickly .
Extractive summary	another embodiment for the provision of the citric acid solution 30 in the first compartment 14 is illustrated in fig4 and is particularly useful when the fluid in the container means is a carbonated beverage as the generation of the carbon dioxide gas continues will provide space for further expansion of the expandable pouch means 2 . the container 40 is illustrated as being a tube but it is to be understood that it can be of any desired geometrical configuration .
Abstractive summary	An expandable pouch means comprising two relatively flat sheets of a flexible plastic material in superposed relationship and formed into a first compartment and a plurality of other compartments by a plurality of lengthwise extending strips which join together opposed portions of the flat sheets using a semipermanent pressure- rupturable sealing means.

Table 7: Summarization example 2