Automatically Inferring the Document Class of a Scientific Article

Antoine Gauquier
antoine.gauquier@ens.psl.eu
IMT Nord Europe & Télécom Paris
& DI ENS, ENS, CNRS, PSL University & Inria
Paris, France

Pierre Senellart
pierre@senellart.com
DI ENS, ENS, CNRS, PSL University
& Inria & IUF
Paris, France

ABSTRACT

We consider the problem of automatically inferring the (LaTeX) document class used to write a scientific article from its PDF representation. Applications include improving the performance of information extraction techniques that rely on the style used in each document class, or determining the publisher of a given scientific article. We introduce two approaches: a simple classifier based on hand-coded document style features, as well as a CNN-based classifier taking as input the bitmap representation of the first page of the PDF article. We experiment on a dataset of around 100k articles from arXiv, where labels come from the source LaTeX document associated to each article. Results show the CNN approach significantly outperforms that based on simple document style features, reaching over 90% average F_1-score on a task to distinguish among several dozens of the most common document classes.

CCS CONCEPTS

• **Information systems → Information extraction**.

KEYWORDS

scholarly articles, information extraction, image classification, document class, PDF

ACM Reference Format:
Antoine Gauquier and Pierre Senellart. 2023. Automatically Inferring the Document Class of a Scientific Article. In *ACM Symposium on Document Engineering 2023 (DocEng '23), August 22–25, 2023, Limerick, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3573128.3604894

1 INTRODUCTION

The majority of research papers in fields such as mathematics, physics and computer science are written using the LaTeX document composition system. LaTeX documents have a *document class*, which defines the type of document to be generated and how it is styled. The standard LaTeX document classes include `article`, `book` and `report`, but many others have been defined and are included in modern LaTeX distributions. In particular, many publishers of academic journals and conference proceedings created specific document classes, to define their own document structure standards, and to get a uniform style for all the papers in a given conference or journal. For instance, the current paper was generated using the `acmart` document class, which is commonly used to write articles for venues sponsored by the Association for Computing Machinery. Each document class structures papers differently: some are formatted in single-column, others in double-column; different margins or fonts are used; author names, abstracts or page numbers are displayed differently.

Given a scientific article in PDF format, determining its document class has several applications. First, systems for information extraction over scholarly articles (such as GROBID [14] for meta-data extraction or TheoremKB [15] for extraction of mathematical statements) could benefit of knowing the document class to train better models: it is easier to know how to extract section headings, author's institutions or statements of theorems when the document class if known and all these elements are formatted similarly from one article to the next. Second, academic search engines such as Google Scholar[1] or BASE Search[2], which index PDF articles found in preprint repositories and on the Web at large could derive information about where a document was published (e.g., that it is published at an ACM or IEEE venue) based on its document class.

However, determining the document class of a scientific article is far from trivial. Indeed, we often only have access to a PDF version of the paper, and not to its LaTeX source code. It is true that it may be easy for a human being familiar with the various famous document classes to determine, given only the PDF of the paper, the document class used. However, this manual method cannot be scaled up to the use cases above. This motivates the current work, which explores automatic inference of the document class of a given scientific article in PDF.

There is a relatively rich literature on information extraction from scholarly articles. For instance, there is previous work on extraction of headers and meta-data [1, 6, 14], citations [19], acknowledgments [11] or figure meta-data [3].[3] The exploitation of the layout and visual rendering of PDF documents to make inference about their content or structure has also been considered [10, 22, 23], especially for applications such as extraction of data from invoice-type documents. However, to the best of our knowledge, the specific task of LaTeX document class inference from PDF articles has not been addressed to this date.

The goal of this work is to propose relatively simple, scalable, tractable, and effective methods to achieve this classification task. We propose a supervised machine learning approach to this classification problem, each class corresponding to one (or several related) document class(es). A first idea is to engineer discriminant features

DocEng '23, August 22–25, 2023, Limerick, Ireland

ACM ISBN 979-8-4007-0027-9/23/08…$15.00
https://doi.org/10.1145/3573128.3604894

[1] https://scholar.google.com/
[2] https://www.base-search.net/
[3] More examples can be found on the CiteSeerX webpage https://csxstatic.ist.psu.edu/downloads/software

based on some simple geometric and style-based characteristics of the document. A second idea is to rely on the rendering of the PDF document as a bitmap image and use standard modern image classification techniques. Indeed, these are the two methods described in this paper: using a set of simple, hand-coded document style features; and using computer-vision methods relying on deep learning.

In Section 2, we present the methodology used to construct a set of five, simple, hand-coded document style features, based on human knowledge of the structuring of scientific publications. In Section 3, we introduce a more advanced model based on convolutional neural networks (CNNs) [7, 12]; we also identify the presence of so-called *heterogeneous* classes of documents, and the interest of using reject options [9]. In Section 4, we discuss our dataset, as well as performance metrics used to analyze our results. Finally, we presents experimental results for both approaches in Sections 5 and 6, before concluding with perspectives for future work.

The code, trained models, and instructions to obtain the dataset can be found at https://github.com/AntoineGauquier/inferring_document_class_of_scientific_article/. Additional content (especially a list of all labels and detailed experimental results) is available in an extended version of this paper, available from the same repository and as supplementary material.

2 CLASSIFICATION ON HAND-CODED FEATURES

In this section, we discuss the method developed to infer the LaTeX document class with only five, sample, hand-coded, document style features.

In order to construct our set of features, we generate an XML representation of each paper, thanks to a tool called pdfalto[4], which automatically reconstructs a structured line-based representation of a PDF document. More precisely, pdfalto is a tool for parsing PDF files and producing an XML representation of the PDF content in the ALTO[5] standard format, with information about page geometry, individual text tokens and lines, font styles, etc.

2.1 Hand-Coded Document Style Features

Before going into the technical details of the definition of features, let us motivate our choice of variables. We follow two main goals. First, we want to investigate whether a classification approach is relevant, i.e., that feature values have different distributions per different document class and thus make the data separable. Second, we want to validate that an approach based on geometry is relevant, which paves the way for the CNN model presented in Section 3. This is why we chose to construct a set of five simple, human-understandable, geometric features. The idea of each feature took birth through human observation of scientific papers, and through the prior knowledge that we have about how scientific papers are usually organized.

We thus want to exploit: left margins of text blocks (1), top margins on each page (2), the width of text columns (3), as well as font families and font sizes used. A graphical illustration of the first three of these elements is shown in Figure 1.

[4]https://github.com/kermitt2/pdfalto
[5]https://www.loc.gov/standards/alto/

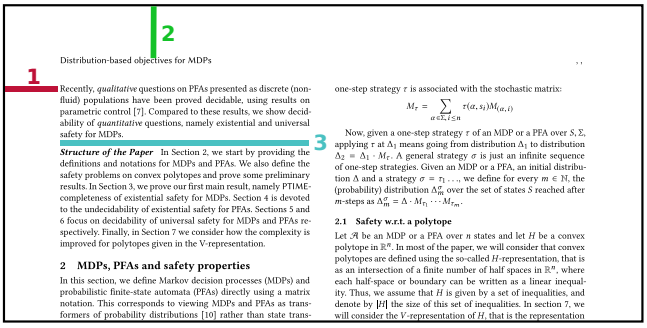

Figure 1: Illustration of major geometric elements: 1/ left margin; 2/ top margin; 3/ column width

However, each of these elements is presented in multiple copies in the document. Indeed, a scientific article almost always include several blocks of text, several pages and several fonts. Therefore, it is necessary to build aggregated features composed of the different instances of the elements presented above. We now present our five features.

Weighted average left margin (lm). The average is weighted by text-block length, in order to obtain the most representative value for the left margin. Let us denote m^{h}_i the left margin of the i-th text-block of the document and l_i its vertical height. Then, the weighted average left margin $\overline{m^{\mathrm{h}}}$ is defined as:

$$\overline{m^{\mathrm{h}}} = \frac{\sum_{i=1}^{N_b} m^{\mathrm{h}}_i \times l_i}{\sum_{i=1}^{N_b} l_i}$$

where N_b denotes the number of text-blocks in the document. This is a numerical feature (floating-point number).

Average first top margin (tm). We are here interested in the gap between the top of the page and the very first block of content (and not between the top of the page and all blocks of content). By denoting $m^{\mathrm{v}}_{i,j}$ the distance between the top of the page and the j-th block of the i-th page of the document, the average first top margin $\overline{m^{\mathrm{v}}}$ is defined as:

$$\overline{m^{\mathrm{v}}} = \frac{\sum_{i=1}^{N_p} \min_j m^{\mathrm{v}}_{i,j}}{N_p}$$

where N_p denotes the number of pages in the document. Again here, this feature is numerical (floating-point number).

Weighted average column width (cw). Technically, it corresponds to the average of the width of every text block in the document, weighted by the height of the block (so as not to put importance on very short blocks, such as equations or sections headings, which do not have a typical width). We here denote w_i the width of the i-th text-block in the document, and l_i its vertical

height. The weighted average column-width $\overline{w}$ is defined as:

$$\overline{w} = \frac{\sum_{i=1}^{N_b} w_i \times l_i}{\sum_{i=1}^{N_b} l_i}$$

where N_b is still the number of text-blocks in the document. Note this is very similar to the definition of **lm**. It is also a numerical feature (floating-point number).

Most common font family (ff). Choosing the most common font family requires defining a criterion. We choose to quantify the importance of a font family by the space it occupies in the document. Assume we have N_f different fonts used in the document (in the ALTO representation, this corresponds to the number of `<TextStyle>` tags). Let us denote S_i the set of all tokens in the document styled in the i-th font of the document. We define the font importance f_i of the i-th font as:

$$f_i = \frac{\sum_{s \in S_i} l_s \times h_s}{\sum_{j=1}^{N_f} \sum_{s \in S_j} l_s \times h_s}$$

where l_s is the length of token s, and h_s the height of token s. The product $l_s \times h_s$ thus gives an area, and we compute the space ratio that each font family occupies. To finally get the most common font-family f^{family}, we take the font family of the font with the highest f_i:

$$f^{\text{family}} = \text{family}\left(\arg \max_{i \in \{1,\ldots,N_f\}} f_i\right).$$

This feature is a categorical one.

Most common font size (fs). The computation is based on the computation f_i defined above for each font i, and then we obtain the feature

$$f^{\text{size}} = \text{size}\left(\arg \max_{i \in \{1,\ldots,N_f\}} f_i\right).$$

This is a numerical value, though in practice it mostly acts as a categorical feature as the number of different font sizes is limited.

2.2 Random Forest Model

We train a supervised machine learning model based on these five features per document. After trying different models such as Support Vector Machines (SVM) [4] or logistic regression, we settled on using random forests [2]. Random forests have the advantage of giving good classification results on this kind of task (as we experimented compared to SVMs and logistic regression), are reasonably fast to train, and lend themselves well to explanations.

In more detail, models based on random forests are a type of ensemble methods. The principle of ensemble methods is to combine multiple weak learners (i.e., models whose performance is not considered as being good enough) in order to have a global model with good performance. The weak learners of random forests are decision trees. On each node of the tree, a condition (usually, an inequality) over one feature of the data is evaluated. Depending on the answer, a branch is followed, and either leads to another node (and thus, another evaluation), or a leaf, corresponding to one of the possible classes. The random forest is thus composed of a set of decision trees (also called estimators), each of them being trained with a different subset of the data. The random forest provides the data to be classified to all the estimators, and finally makes a majority vote to decide which class is to be selected.

3 CNN-BASED APPROACH

In this section, we present an alternative approach, based on convolutional neural networks, trained on a bitmap rendering of the PDFs. We first explain which bitmap images are selected as input to the model. We then present the general composition of a CNN, as well as a sample simple architecture. We subsequently describe other standard architectures we train, as well as their complexity through two criteria. Finally, we discuss a limitation of the modeling with respect to heterogeneous document classes, as well as a potential solution.

3.1 Input to the Model

In order to use CNNs on PDF renderings, we need to decide which exact content to show to the model. A natural idea is to simply transform each page of the PDF into an image, by choosing a fixed resolution and size for each image, so that the input size of the model is the same for all papers, no matter the original format of the paper. This of course has for consequence to distort the aspect ratio of the PDFs, but it still preserves their structure and organization on the page.

Convolutional neural networks are particularly suited for our task, since they perform well in image processing and geometric pattern learning through the use of convolution, while limiting the number of parameters to be trained (in comparison with fully connected, feed-forward, neural networks). However, they do not support the observation of sequential data (in our case, a sequence of images from the PDF paper) as RNNs or transformers do, so we decided to only keep one image per paper, namely the rendering of the very first page of the paper, since it contains much information that can be used to discriminate document classes: position and size of the title, of the author's names, of the abstract, of copyright statements, etc. Some document classes even have some specific content in a header or footer.

An example of such an image, at the exact resolution that we use in experiments and which is taken randomly from our dataset, is shown in Figure 2. Note that, though the text is illegible, much information about the structure, geometry, and style of the document, which are helpful to determine the document class, is apparent.

3.2 Design of a Simple Architecture

Convolutional Neural Networks [7, 12] are a deep learning technique involving an input layer, an output layer, and at least one hidden layer. As opposed to fully connected layers of other forms of neural networks, CNN uses *convolutional layers*, based on the *convolution* operation. Denoted $*$, it is defined, in its discrete form in two dimensions, which is the one we are interested in, as:

$$(f * g)_{m,n} = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f_{i,j} \times g_{m-i,n-j}$$

Figure 2: Example of a generated image from a paper, taken randomly in the dataset

where f is the kernel or filter, and g is the 2D input. For numerical applications, f and g are finite.

The parameters of the model consists in the values of the f matrices. We can have several filters per hidden layer, as well as several hidden layers. In addition to these convolution operations, we also apply a set of other operations, such as: activation functions, to know which convolutional neurons are activated by the parameters; pooling operations, to progressively reduce the size of the images through the hidden layers and to get a small set of embeddings at the end of the model; and dropout operations, which aim at randomly dropping some of the neurons to avoid overfitting.

We then concatenate several of these layers, and finally add a flattening operation and one (or several) last fully-connected layer, to transform our set of small matrices of embeddings into a set of *logits*, corresponding to the number of classes in our problem.

As a sample simple architecture (but see following section for more complex ones), we designed, in a fairly typical way for simple CNNs, a simple model with four hidden layers, with increasing numbers of filters and decreasing size of the embeddings throughout the layers. Its architecture is presented in Figure 3.

3.3 Comparison with Other Architectures

Our assumption is that our simple architecture is sufficient to discriminate document classes since, as already mentioned above, first images of papers contain an important number of possible patterns indicative of document classes. To verify this assumption, we also train some more complex architectures from the state of the art. Thus, we also consider the following ones: a 50-layer ResNet from 2016 [8], a Mobile version of NASNet from 2018 [24], and a B0 version of EfficientNetV2 from 2021 [20]. These are chosen because they are standard CNN architectures and have a reasonably limited number of parameters. We pay special attention to EfficentNetV2-B0, first because it is the most recent of the considered architectures, and second because it was designed to propose better parameter efficiency than previous models. We set aside even more modern architectures with a large number of parameters, such as the ConvNeXt architecture of 2022 [13], whose basic model nearly reaches 90 million parameters.

Table 1: Complexity comparison between different architectures of CNN, for input size 256×256

Architecture	Total # of parameters	FLOPs
Our simple architecture	0.04M	1.36B
ResNet50V2	23.63M	9.13B
NASNetMobile	4.30M	1.50B
EfficientNetV2B0	4.09M	0.80B

To be able to compare the complexity of these different models, we present in Table 1 the total number of parameters of each architecture, as well as the number of floating-point operations (FLOPs) required for inference (as defined in [21]), for the same (fixed) size of input images, which is a machine-independent way to measure the time for inference.

It is important to consider these two criteria in order to check that our model is respectively simple, light (thus, that its number of parameters is limited and that it takes little space in memory), as well as fast, in training phase, but especially in inference phase: it is an essential criterion so that the model can be scaled up.

3.4 Modeling with Reject Option

As mentioned in the introduction, LaTeX includes several standard document classes (including `article`, `book`, and `report`). In fact, these are still widely used because they are guaranteed to be installed on any LaTeX distribution, and because authors are used to them. Indeed, it is very common to find articles with widely different styles that have been written using these classes (chosen "by default") and numerous extra LaTeX packages to customize the style. This is in contrast with document classes developed by publishers, which are usually followed quite strictly.

We can thus expect some difficulties in determining the document class of a paper, when there is too much heterogeneity in documents produced using them. As an alternative to the modeling proposed so far, we consider an approach allowing us to put apart these so-called *heterogeneous classes*, in order to potentially improve the performance of our classifier on non-heterogeneous classes.

The idea of judging a subset of a dataset as "irrelevant" for a machine learning model is something that has already been studied under the name of classification with *reject option* [9]. The name comes from the idea that we want our model to be able to reject a prediction (or a training sample, considering training phase) if it belongs to the irrelevant subset of our data (of course with respect to the classification task we are considering).

There are two main kinds of reject options: novelty rejection and ambiguity rejection. In the first case, the model receives an

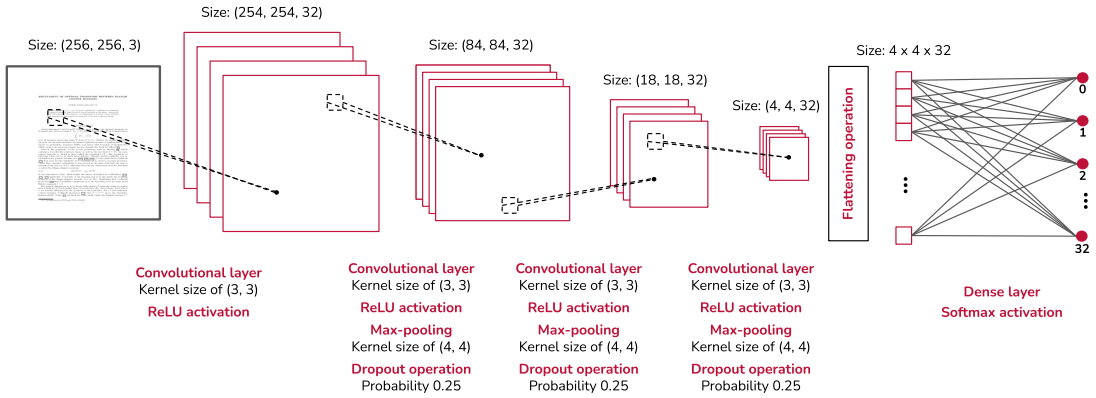

Figure 3: Diagram of our CNN architecture

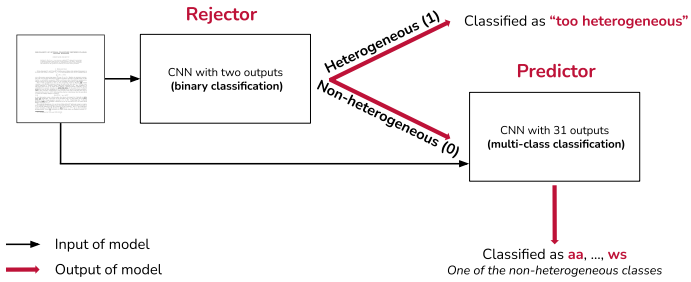

Figure 4: Modeling with reject option

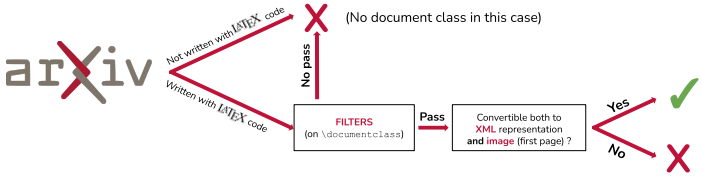

Figure 5: Selection process of papers

unobserved sample that is significantly dissimilar to what has been seen during training phase; in the second case, the learned model does not behave correctly in certain regions of the instance space it is evolving in. The idea of separating the heterogeneous classes from the other ones is closer to the first type of rejection: the model is observing, in inference phase, papers associated to one of the heterogeneous classes having a structure which is too dissimilar with the ones of this class on training phase. However, it is also linked to ambiguity rejection, because a paper with completely personalized structure from a heterogeneous class could be equally distant from several document classes.

In order to keep a simple modeling, and since we expect the task of distinguishing heterogeneous classes to be hard, we stick to novelty rejection, and we propose an architecture that uses a separate *rejector*, i.e., having one dedicated model for rejection, and one dedicated model for classification. The samples that are classified by the rejector as too heterogeneous are not presented to the classifier. A diagram showing this overall process is presented in Figure 4. The architectures of each of the models (both rejector and classifier) are close to the one presented in Section 3.2: we only modified the last dense layer to be consistent with the tasks we do (respectively, binary classification and multi-class classification with 31 possible classes). We also added one dense layer for the rejector before the last dense layer to help it discriminate heterogeneous classes.

4 DATASET AND PERFORMANCE METRICS

In this section, we present the dataset we are using to conduct our experiments, as well as some (usual) performance metrics used to evaluate our trained models.

4.1 Dataset

To conduct our experiments, we used a set of 119 087 scientific papers, extracted from the ArXiv[6] platform, a free open access archive of scientific publications. This set of papers corresponds to all papers from the year 2018. The choice of this platform is justified by the joint availability of the PDF document and its LaTeX source, both of which can be obtained through arXiv's bulk access from AWS[7]. The document class is extracted from the LaTeX source through simple rules based on regular expressions. However, among the 119 087 papers, only 98 713 could be exploited. Indeed, for some of them, the LaTeX source is missing, for example in the case where the paper was written using word processing software such as Microsoft Word, making it impossible to infer any document class. For others, document class extraction was ambiguous or impossible (e.g., some authors use Plain TeX instead of LaTeX or redefine the `\documentclass` macro), or the extraction of ALTO or image representations was not possible (because of failures of pdfalto or of the bitmap rendering); for more on arXiv extraction issues, see [5, Section 6.2]. A diagram summarizing these steps is shown in Figure 5.

[6] https://arxiv.org/
[7] https://info.arxiv.org/help/bulk_data_s3.html

Within these remaining 98 713 documents, we identify more than 1 200 different document classes (based on their names). But we observe that an overwhelming majority of papers correspond to only a few classes. Indeed, the first 20 classes represent 90% of the papers, and more than half of document classes are only used in one paper. In order to be able to use statistical learning methods, we need enough data for each class so that the models can be properly trained. We therefore impose a criterion of data representativeness, namely that each class kept must contain at least one thousandth of the data (thus appearing in around one hundred articles). 44 document classes satisfy this criterion. As `article`, the most commonly used class, was also the one that was the most heterogeneous, we renamed that class to `other` and mapped all documents whose class was not among the 44 to that class. This means the `other` class partly act as a reject option, though we also consider a better rejection strategy as detailed in Section 6.

The reduced list of 44 classes still contain very similar classes (as can directly be observed by name similarities). For example, classes `aastex6`, `aastex61` and `aastex62`; `ieeeconf` and `ieeetran`; or `amsart` and `amsproc`. It is therefore likely that some classes are in fact very similar, but a similarity in name only is not enough to justify merging these classes. We could also miss similar classes whose names are too different to predict that they are similar. To properly deal with similar document classes with different names, we compute a similarity metric between the source codes of the `.cls` document classes, usually available within the arXiv source archive.

We chose term frequency-inverse document frequency (TF-IDF) [18], which evaluates, for a given document class, the importance of each term in the source code of that class. We therefore compute a vector whose size is the number of distinct terms and associates to each term an importance score. Once those scores are computed for all document classes, we can compute a pairwise similarity, by computing, for all pairs of document classes (i, j) with $i \neq j$ the dot product of their TF-IDF vectors. This results in a score between 0 and 1, representing how similar the pair of document classes is. We set a threshold of 0.8.

By applying this rule, we have a new short-list of 33 classes, corresponding to individual or groups of document classes, which is our final label set for the different approaches.

We split our dataset between a train and a test set, by randomly selecting papers for each class with a ratio of 0.8 (80% of the dataset for training, and 20% of the dataset for test).

The dataset is heavily unbalanced across document classes: the biggest class `other` has 31 759 samples, while some others have just above 100 instances. To deal with this while training our models, we use oversampling techniques [16]. For each of the 32 classes except `other`, we randomly duplicate papers to reach the same number of samples as class `other`. We do this within the training set so as to do unbiased training across classes; this allows the model to pay as much attention to each of the classes, but also to show more data to the model, thus ensuring its convergence and stability. We also perform oversampling over the test set, in order to properly measure its performance, as described next.

4.2 Performance Metrics

We define performance metrics to evaluate and compare our results on the test set. Global accuracy of the classification over the entire test set is not sufficient as we want to investigate in more detail misclassification errors. We therefore report the individual precision, recall, and F_1-score of the classification for every class (in the extended version of this paper) and their mean (as a summary), which will be unaffected by data imbalance.

Let us denote TP_i the number of "true positives" for class i, i.e., the number of samples that the model classifies in class i and that actually belong to this class. TN_i will then be the number of "true negatives" for class i, i.e., the samples that were rejected by the model (for a multi-class problem, it corresponds to the situation where a model classified the sample **not** to belong to class i, so basically to any other class), while actually not belonging to this class i. We also define FP_i as the number of "false positives" for class i, which is the number of samples that the model classifies in class i while belonging to another class. Finally, we define FN_i the number of "false negatives" for class i, which correspond to the samples that are rejected by the model for class i but that do belong to this class i.

The precision score of a given class i is defined as follows:

$$\text{precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i}$$

The recall score of a given class i is defined as follows:

$$\text{recall}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i}$$

And the F_1-score, which is the harmonic mean on of both precision and recall, is defined as follows:

$$\text{F}_1\text{-score}_i = 2\frac{\text{precision}_i \times \text{recall}_i}{\text{precision}_i + \text{recall}_i}$$

Note that per-class recall is the same as per-class accuracy: the proportion of documents of this class that are correctly classified.

Precision, recall, and F_1-score are then summarized across all classes by their unweighted arithmetic mean.

5 EXPERIMENTS FOR HAND-CODED FEATURES

In this section, we present the data distribution and experimental results obtained with the five hand-coded document style features described in Section 2 and the random forests model built on top of them.

5.1 Data Distribution

We analyze the distribution of feature values on the entire dataset of 98 713 articles described in Section 4. Table 2 presents some key figures about the distribution of the data, for all the numerical features (i.e., all except **ff**).

Weighted average left margin (lm). Figure 6 shows the plot of the histogram for this feature. As with all other histogram charts (Figures 7 to 9), the x-axis represents the values taken by the feature (i.e., here, the value in points[8]), and the y-axis the corresponding frequency (as a number between 0 and 1, summing up to 1) of the

[8]The *point* is the space unit in the PDF file format, with value $\frac{1}{72}$-th of an inch.

Table 2: Key figures of the distribution of the numerical hand-coded features[8]

Feature	Unit	Mean	Median	SD
lm	points	159.99	166.42	38.20
tm	points	89.58	87.39	44.33
cw	points	424.25	228.51	65.93
fs	size	10.44	10.00	0.89

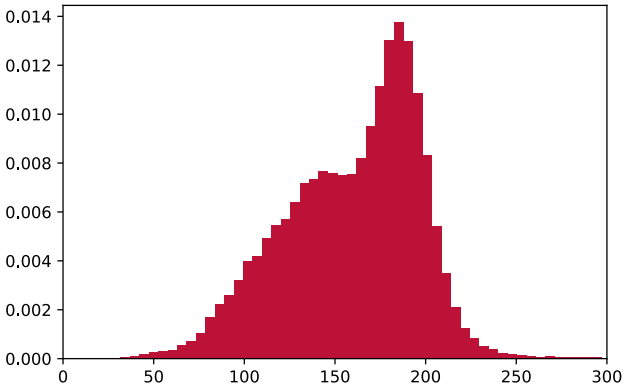

Figure 6: Histogram of values of weighted average left margin (lm)

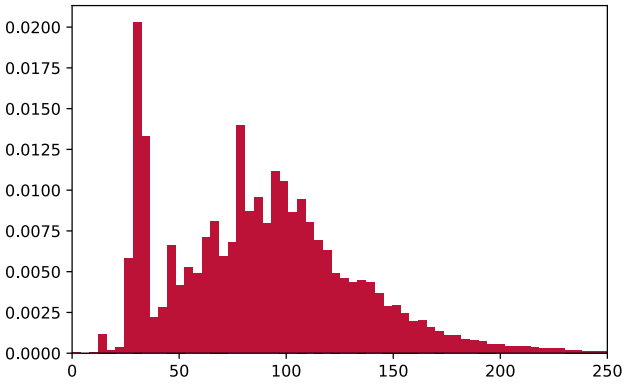

Figure 7: Histogram of values of weighted average first top margin (tm)

range values referenced by the histogram bar. We observe what seems to be the superposition of two Gaussian-like distributions: the first with a mean close to 140 and the other one with a mean close to 190 (some common document classes such as `llncs` or `article` when unmodified indeed have a high left margin).

Average first top margin (tm). Figure 7 shows the plot of the histogram for this feature. We can also observe a Gaussian-like distribution of the data, around the value 90, with an important peak at approximately 40 points.

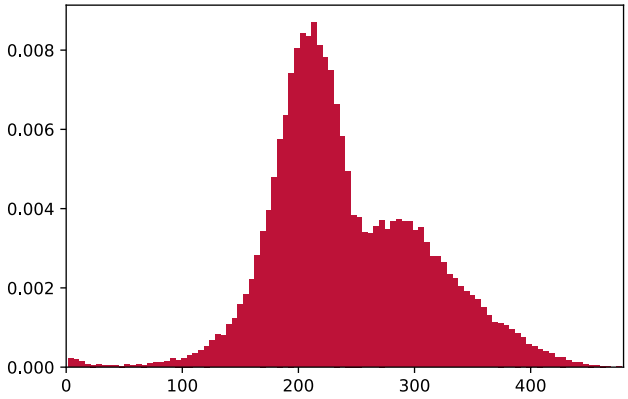

Figure 8: Histogram of values of weighted average column width (cw)

Weighted average column width (cw). Figure 8 shows the plot of the histogram for this feature. By observing the shape of the histogram, we again identify what seems to be the superposition of two Gaussian-like distributions: one with mean around 200 points, and one around 300 points. Given the fact that the width of a page is usually around 600 points[9], and that some space is kept for margins, these two distributions most likely respectively correspond to papers with two columns (smaller column widths) and one column (larger column widths).

This tends to show that the column-width represents papers with two or one columns: the papers with two columns are more likely to have smaller horizontal margins than papers with a single column.

Most common font family (ff). This feature is different from the other ones, since it is categorical. We cannot describe it with continuous, numerical metrics, but the histogram presented in Figure 9 is sufficient to correctly describe its distribution. It is important to note that we have not displayed all possible font-families, since there are 109 of them, and that would have become impossible to read. In the histogram, we only show font families that have at least 100 representatives (samples that take this value). The rest is put together in a class "All others". We see that we only have a few significant font families in terms of frequency (the three most frequent being `cmr`, i.e., the default Computer Modern font family of TEX; `nimbusromnol`, i.e., Nimbus Roman No.9 L, a free serif font designed to have the same metrics as Times New Roman; and `sfrm`, the Computer Modern font from the CM-Super family which is a re-design of TEX's classic Computer Modern font). Other font families may still be important to identify specific document classes.

Most common font size (fs). Figure 10 shows the plot of the histogram for this feature. We observe here that, despite the fact that it is a numerical feature, we could consider it a categorical one. This is due to the fact that the font sizes associated with the most common fonts are usually 9, 10 (TEX's default font size), 11, or 12 points, with very few exceptions.

[9] More precisely, 612 points for US letter and approximately 595 for ISO 216 A4.

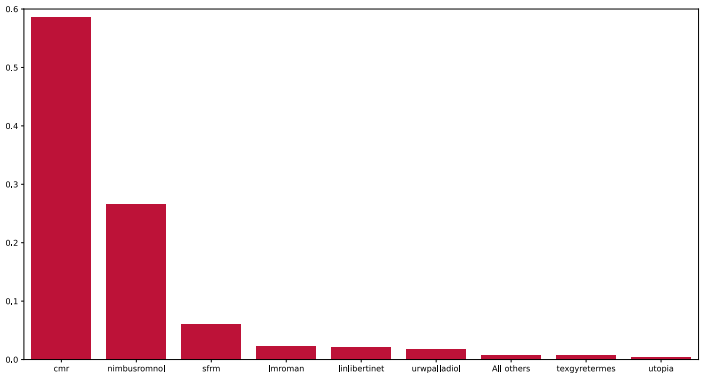

Figure 9: Histogram of values of most common font-family (ff)

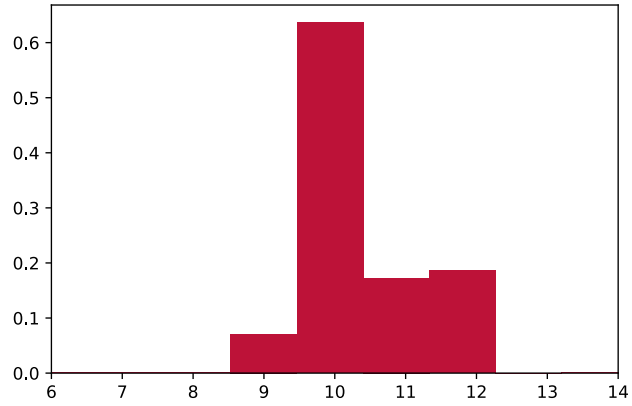

Figure 10: Histogram of values of most common font-size (fs)

Table 3: Correlation matrix, involving *numerical* hand-coded document style features

	lm	tm	cw	fs
lm	-	-20.46%	-65.28%	-1.86%
tm		-	9.62%	1.17%
wc			-	3.22%

Correlation of features with each other are presented in the Table 3. Note the strong (negative) correlation between the **cw** and **lm** which is intuitive: a document with small column width will have high left margin, for a fixed number of columns.

5.2 Results of the Random Forest Model

We now report on the results of the random forest model. The use of random forests in practice requires making some choices in terms of hyperparameter. The first one to be considered is the minimum number of samples per leaf in the decision trees. In fact, since we have important heterogeneity in our data, and given the quite high number of classes, there is a high risk of overfitting if we let this parameter free (i.e., no minimum number of samples per leaf: we could end up only having one sample per leaf). In fact, without setting this one, we ended up having 100% of accuracy on the training set (classes being now balanced), but low global accuracy on the test set (around 40%). We set this parameter to be 0.01% of total observed samples. The second such parameter is the number of estimators (i.e., the number of decision trees involved in the forest). We set it to 1 000; the default value is 100, but this would low for a 33-class classification process and might impact the stability of the most common decision. We used the random forest implementation of the scikit-learn [17] Python library.

By training such a model with these parameters, we got, on the training set, a global accuracy of 87.6%, and of 66.0% on the test set. The mean precision is 64%, recall 66%, and F_1-Score 64%, all on the test set.

Note that a *Dummy* classifier which would predict the most frequent class would have an accuracy of $\frac{31\,759}{98712} \approx 32.2\%$ while it would have a mean recall of $\frac{1}{33} \approx 3.03\%$, a mean precision of $\frac{32\times 0+1/33}{33} \approx 0.09\%$, and F_1-score of $\approx 0.18\%$. This illustrates the importance of considering these three metrics instead of just using global accuracy, and this means the random forest model very significantly outperforms the Dummy classifier.

We observe that some classes are really well classified, on which the model generalizes well. We can cite the document class `bmvc2k` which has perfect scores in training, and around 97% in test. We identify other classes well classified by the model, such as `cms`, `eptcs`, `iau/jfm`, `lipics`, `mdpi`, `sigma` or `wbofc`. This shows that the classification approach is relevant for the data we are interested in, since the model succeeds in classifying an important number of classes very well (i.e., the features are such that the data belonging to those classes are easily separable).

But we also identify some classes that are very poorly classified. In particular, two classes have performance in test (for at least one metric) which is similar or even worse than a dummy classifier. Those classes are `other` and `book`. This is not very surprising, as these document classes, as previously discussed are very heterogeneous. Indeed, heterogeneous classes are particularly hard to classify using our random forest model.

Remaining classes have performance that vary from around 25% up to 85% for the different metrics.

Those results are particularly encouraging, since we must keep in mind that we are doing a hard classification task, on 33 classes. Thus, even classes with performance metrics around 25% are way above the results of the dummy classifier. Moreover, this is only using five hand-designed features, chosen from the prior knowledge we have about the domain. This proves that not only a classification approach is justified and relevant for this task, but also that using geometric and style features is relevant.

We consider the *importance* of features used in the random forest model, a measure that quantifies how much each feature on average reduces entropy of the distribution averaged over all nodes of the forest. The feature with highest impact is **tm**, with 32.7% of importance, then **lm** with 19.3% of importance, **ff** 18.4%, **cw** 16.5%, and **fs** has 13.1%. All features thus had some non-negligible impact on the result of the classifier.

6 EXPERIMENTS USING CNN MODELS

In this section, we present the results of the training of the several CNNs we presented in Section 3. We first compare the results of our simple architecture from Section 3.2 with the ones from the literature described in Section 3.3; then we present results related to the modeling with reject option of Section 3.4.

6.1 Results of the CNN Architectures

The training of each CNN model requires specifying some parameters for the image generation part. We recall that each input image has a fixed size. We set this to 256×256. Each image is converted to 256-level grayscale, so each image weighs $256 \times 256 \times 1 = 64$ kB. We report the mean performance (precision, recall, and F_1-score) across all classes, as defined in Section 4.2. The average performance (mean precision, recall and F_1-Score, as defined in section 4) are reported in Table 4, compared to that of the Dummy classifier for context.

Table 4: Mean precision, recall and F_1-Score (over all oversampled classes) for different architectures of CNN

Architecture	Precision	Recall	F_1-Score
Dummy	*0.09%*	*3.03%*	*0.18%*
Our simple architecture	92.67%	92.70%	92.31%
ResNet50V2	93.59%	92.34%	92.28%
NASNetMobile	93.23%	91.17%	91.31%
EfficientNetV2B0	**93.63%**	**93.52%**	**93.43%**

The best performance is obtained with the most recent CNN architecture tested, i.e., EfficientNetV2B0, with 93.4% F_1-Score. However, all models have similar performance, roughly within 1% of the performance of EfficientNetV2B0, all achieving a very high mean precision and recall across classes, significantly above that of the random forest approach with hand-coded features of Sections 2 and 5.

6.2 Results of Modeling with Reject Option

Now that we have trained our CNN architecture with the 33 classes, we can check whether the classes `article` (included in class `other`), `report` (included in class `report/wlscirep`) and book have bad classification performance, because of their heterogeneous nature. Their precision, recall, and F_1-Score on our simple CNN architecture are shown in Table 5.

Indeed, the performance of these three classes is the worst across all classes (the next-worst classes have F_1-score above 87%). This

Table 5: Precision, recall, and F_1-Score for heterogeneous classes on our CNN architecture

Document class	Precision	Recall	F_1-Score
`book`	56.84%	21.39%	31.09%
`other`	69.17%	65.00%	67.02%
`report/wlscirep`	52.09%	77.69%	62.37%

Table 6: Complexity of both rejector and classifier

Model	Total # of parameters	FLOPs
Rejector	0.038M	1.360B
Classifier	0.046M	1.418B
Total	**0.084M**	**2.778B**

Table 7: Precision, recall, and F_1-Score for both rejector and classifier

Model	Precision	Recall	F_1-Score	# classes
Rejector	90.55%	89.15%	89.04%	2
Classifier	96.94%	96.73%	96.83%	31

validates the need for using a rejector to discriminate between homogeneous and heterogeneous classes. First, Table 6 presents the number of parameters and FLOPs for both rejector and classifier. Performance of both rejector and classifier after rejection, as described in Section 3.4 are presented in Table 7.

The performance metrics show that removing the heterogeneous classes from the classifier allows it to perform even better, since we gain 4.5% of F_1-Score in comparison with our modeling including heterogeneous classes. However, the overall pipeline will perform worse than the first CNN modeling, due to the lower F_1-Score of the rejector: even if we had a perfect classifier after rejection, we would still have an overall F_1-Score for the whole pipeline below 90%.

Lower performance does not mean that this modeling is useless. In fact, it may be particularly useful when it comes to processing articles which are known not to belong to heterogeneous classes. Indeed, from Table 8, which presents the detailed performance metrics of the rejector, we see that the recall for non-heterogeneous document classes is above 98%. In practice, it means that less than 2% of papers belonging to non-heterogeneous classes are classified as heterogeneous classes. Since we have a near-perfect recall for the non-heterogeneous document classes, most of them will be presented to the classifier, which has above 96% of F_1-Score, resulting in very high overall classification performance. This can be used in some important use cases. For instance, if we are interested in determining the publisher of a given published (scientific) scholarly article (say, whether it is ACM, IEEE, Schloß Dagstuhl, etc.), it will usually be part of some non-heterogeneous document class; the approach with reject option will allow us to detect outliers while reaching near-perfect classification performance for documents of those publishers.

7 CONCLUSIONS

In this paper, we consider the task of inferring the document class of a PDF scientific article, through the use of classifiers. The first one is a random forest classifier trained on five hand-coded document style features. It is giving satisfactory results considering the number of document classes we are dealing with, but the other model based on computer-vision approaches (through the use of CNNs) gives

Table 8: Performance by class for the rejector, on test set

Class	Precision	Recall	F_1-Score
Non-heterogeneous	83.03%	98.43%	90.07%
Heterogeneous	98.07%	79.88%	88.04%
Dummy (Mean)	*25.00%*	*50.00%*	*33.33%*
Mean	**90.55%**	**89.15%**	**89.04%**

significantly better results, whatever the choice of the precise CNN architecture (a simple one as proposed in the paper or some state-of-the-art architectures). We note that CNNs do require more (GPU) computation power for training and inference than the random forest approach. Moreover, a prediction of the random forest model is relatively easily explainable, and can be related to the decision a human being would make. This is not directly possible with CNNs, which mostly act as a "black box", hard to explain.

The approach proposed in this paper can be applied to a number of tasks, as hinted to in the Introduction, from inferring some publication meta-information from a PDF found on the Web to the use of the knowledge of the document class to improve techniques aiming at extracting information from the PDF of articles. A natural perspective for this work is to assess the usefulness of document class inference for such tasks.

We must keep in mind that the entire study was done on PDFs from a specific dataset: papers published in 2018 on the arXiv which had LaTeX sources available. This means only specific fields, those common on arXiv, are represented, and thus only specific document classes. Also, evolution of document classes before and after that date are not captured. As a consequence, the set of document classes we considered in this paper might not correspond to the set of most common document classes we might encounter in the wild, even only considering scientific articles. We note however that we are unaware of any other labeled dataset with readily available document class information.

ACKNOWLEDGMENTS

This work was funded in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

REFERENCES

[1] Jöran Beel, Bela Gipp, Ammar Shaker, and Nick Friedrich. 2010. SciPlore Xtract: Extracting Titles from Scientific PDF Documents by Analyzing Style Information (Font Size). In *Research and Advanced Technology for Digital Libraries*, Mounia Lalmas, Joemon Jose, Andreas Rauber, Fabrizio Sebastiani, and Ingo Frommholz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 413–416.

[2] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (01 Oct 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[3] Sagnik Ray Choudhury, Prasenjit Mitra, Andi Kirk, Silvia Szep, Donald A. Pellegrino, Sue Jones, and C. Lee Giles. 2013. Figure Metadata Extraction from Digital Documents. *2013 12th International Conference on Document Analysis and Recognition* (2013), 135–139.

[4] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning* 20, 3 (01 Sep 1995), 273–297. https://doi.org/10.1007/BF00994018

[5] Theo Delemazure. 2020. *A Knowledge Base of Mathematical Results.* Master's thesis. Ecole Normale Supérieure (ENS). https://hal.inria.fr/hal-02940819

[6] Huy Hoang Nhat Do, Muthu Kumar Chandrasekaran, Philip S. Cho, and Min Yen Kan. 2013. Extracting and Matching Authors and Affiliations in Scholarly Documents. In *Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries* (Indianapolis, Indiana, USA) *(JCDL '13)*. Association for Computing Machinery, New York, NY, USA, 219–228. https://doi.org/10.1145/2467696.2467703

[7] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press, Cambridge, MA, USA, Chapter 9. http://www.deeplearningbook.org.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 630–645.

[9] Kilian Hendrickx, Lorenzo Perini, Dries Van der Plas, Wannes Meert, and Jesse Davis. 2021. Machine Learning with a Reject Option: A survey. https://doi.org/10.48550/ARXIV.2107.11277

[10] Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. 2022. LayoutLMv3: Pre-training for document ai with unified text and image masking. In *ACM MM*. 4083–4091.

[11] Madian Khabsa, Pucktada Treeratpituk, and C. Lee Giles. 2012. AckSeer: A Repository and Search Engine for Automatically Extracted Acknowledgments from Digital Libraries. In *Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries* (Washington, DC, USA) *(JCDL '12)*. Association for Computing Machinery, New York, NY, USA, 185–194. https://doi.org/10.1145/2232817.2232852

[12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.* 1, 4 (dec 1989), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

[13] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A ConvNet for the 2020s. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11966–11976. https://doi.org/10.1109/CVPR52688.2022.01167

[14] Patrice Lopez. 2009. GROBID: Combining Automatic Bibliographic Data Recognition and Term Extraction for Scholarship Publications, Vol. 5714. 473–474. https://doi.org/10.1007/978-3-642-04346-8_62

[15] Shrey Mishra, Lucas Pluvinage, and Pierre Senellart. 2021. Towards Extraction of Theorems and Proofs in Scholarly Articles. In *Proceedings of the 21st ACM Symposium on Document Engineering* (Limerick, Ireland) *(DocEng '21)*. Association for Computing Machinery, New York, NY, USA, Article 25, 4 pages. https://doi.org/10.1145/3469096.3475059

[16] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. 2020. Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. 243–248. https://doi.org/10.1109/ICICS49469.2020.239556

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[18] Claude Sammut and Geoffrey I. Webb (Eds.). 2010. *TF–IDF*. Springer US, Boston, MA, 986–987. https://doi.org/10.1007/978-0-387-30164-8_832

[19] Guido Sautter and Klemens Böhm. 2014. Improved bibliographic reference parsing based on repeated patterns. *International Journal on Digital Libraries* 14, 1 (01 Apr 2014), 59–80. https://doi.org/10.1007/s00799-014-0110-6

[20] Mingxing Tan and Quoc Le. 2021. EfficientNetV2: Smaller Models and Faster Training. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 10096–10106. https://proceedings.mlr.press/v139/tan21a.html

[21] Raphael Tang, Ashutosh Adhikari, and Jimmy Lin. 2018. FLOPs as a Direct Optimization Objective for Learning Sparse Neural Networks. In *CDNNRIA*. https://arxiv.org/abs/1811.03060

[22] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. LayoutLM: Pre-training of text and layout for document image understanding. In *SIGKDD*.

[23] Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, et al. 2021. LayoutLMv2: Multi-modal pre-training for visually-rich document understanding. In *ACL/IJCNLP*.

[24] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 8697–8710. https://doi.org/10.1109/CVPR.2018.00907