Document Classification in Public Administration

Gugole Nicola
Pasquali Alex
Piccoli Elia

September, 2021

Final project for the
Human Language Technologies
course

University of Pisa
Artificial Intelligence
A.Y. 2020/2021

Contents

1 Project Purpose

The project was developed in association with *Compagnia Trasporti Toscana (CTT)*, which is the company that handles the public transportation in Tuscany, and Doc. Riccardo Franchi (Corporate Manager), who was our reference inside the company. The focus of the project was to improve the document archiving process in compliance with the *Protocollo Informatico Italian* law. In fact, the user of public administration, in order to archive a protocol, has to fill various fields (**Figure 3**), in particular: needs to write a summary of the contents of the document (*Oggetto*), pick the correct document repository for the office (*Contenitore*) and then choose the correct class among many. The different classes are stored in a hierarchical structure (*Titolario*, **Figure 4**), where going from higher to lower levels a more precise description of the class is provided. The user, in order to select the class, has to navigate through the various levels of the hierarchy. Each office is responsible for the administrative processes identified by the classes of its competence. This results in a mechanic and repetitive task, that the user must do each time a new protocol arrives. Obviously, this process is not error-free since it is heavily influenced by the user competence, each error affect the quality of the administrative process.

The purpose of the project is to speedup and provide soundness to the class selection task providing the user with a suggestion formed of a small set of classes where the document is more likely to be classified. In this way, without going through the hierarchy, the user is able to exploit the model prediction. In order to properly build the model architecture a key point was to define input and output. The *output* can be easily defined as a distribution probability over the different classes. The *input*, instead, was limited to only a small subset of available information. As a matter of facts, the input is composed by two elements: *Oggetto* and *Contenitore*. **Section 3.1** will provide more information on how the two different fields are handled. An important aspect that will play a key role is the *inference time*, in fact the model needs to be fast in order to give an almost instant feedback to the user that would otherwise need to search among the hierarchy.

The analysis will be divided into different sections. Starting from an in depth analysis of the dataset in **Section 2**, followed by the analysis of the two main task of the project. **Section 3** will cover the easier task using only

the first level of the classes hierarchy resulting in a probability distribution of *15* elements. **Section 4**, instead, will focus on the second level of the tree leading to *118* possible classes. The remaining sections will analyze different architectures, the results over a new set of data and some final considerations.

2 Dataset Preprocessing

CTT company kindly offered us its entire dataset, with the hard constraint of taking the data starting from January 2018 because of a change of class tree which happened at the time. The raw dataset consisted of **100000** samples and needed a meticolous preprocessing to avoid privacy issues. Further parsing was also needed to generalize otherwise futile vocabulary entries.

2.1 Dataset Features Extraction

The dataset was composed initially of many features comprising date of insertion, object, office, class code, class name and various ids as external keys to other dataset tables.
Of these only a subset was kept for the model input, as explained in **Section 1**. The dataset is therefore composed of 4 features:

- **Oggetto:** string defining a summary of the document which is being classified.
- **Contenitore:** string representing the document repository where the protocol is archived. From now on, for simplicity sake, the document repository will be referred to as *office*.
- **PrimoLivello:** integer ranging from 1-15, ground truth for first level classification.
- **SecondoLivello:** integer with a value range depending on the first level (each first level may have a different subtree), combined with *PrimoLivello* gives a ground truth for second level classification.

2.2 Dataset Cleaning

A series of operations was applied in order to privatize and generalize the data as much as possible, trying at the same time to keep highest possible

variability for better learning.

Exploring the samples and discussing with Doc. Franchi lead to the individuation of personal data as well as offices for which the task is useless. Two situations in particular lead to a first skimming of the dataset:

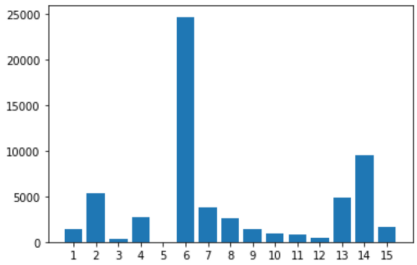
- A specific class (*Permessi sindacali*) contained a notable amount of full names, and was therefore dropped under Doc. Franchi advice.
- An ensemble of protocols all involving *Fatture* was dropped due to the uselessness of a classifier in that case. The documents involving these protocols are in fact not inserted by hand but by an automatized process.

This first process left us with a dataset composed of **60509** samples, which we furthered cleaned and treated by transforming the input features (*Oggetto* and *Contenitore*) into lower case strings and by stripping the resulting samples from any unnecessary leading/trailing whitespace.

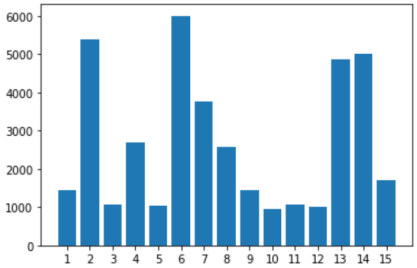
2.3 Dataset Balancing

The implicit different load of documents per class in a real world application such as this one leads to a not surprising fact, the unbalancing in number of samples per class. An unbalanced dataset is not ideal for the task, since the model will have a disproportion of learning material with a consequent lower precision in less populated classes.
We therefore opted for a balancing process on the dataset, applying a careful stratification to low populated classes and a reduction to exaggeratedly populated ones. The main aim of the process was to result in a more balanced first level dataset (the second level of the tree is too unbalanced and sparse) while maintaining the data distribution.

- **Stratification:** applied to class 3,5,11 and 12. Unfortunately we could not automatically generate new samples in the *Oggetto* field because of its textual nature, nevertheless we tried to higher the number of samples and their variability by creating new rows where *Oggetto* and *Contenitore* are chosen independently and randomly from the pool of *Oggetti* and *Contenitori* of a specific second level class. This augmentation assured the correctness of the dataset both for the first level and the second level task.



(a) Before balancing



(b) After balancing

Figure 1: Class distribution

- **Reduction:** applied to class 6,14. Randomly chosen samples were dropped. Randomization was preferred to cutting beginning/end of dataset for a higher probability in maintaining the interclass distribution.

The result of the balancing process can be appreciated in **Figure 1a** and **Figure 1b**.

2.4 Dataset Parsing

Data parsing is a fundamental aspect when working with Natural Language, therefore we applied a last series of filtering and mapping to our data before feeding it to the model. Because of the predetermined and fixed range of values for *Contenitore*, we decided to apply the parsing only to the *Oggetto* field, definitely more dynamic than the other. The treating of *Contenitore* will be further discussed in **Section 3.1.2**.
The dataset has therefore to pass through a last procedure before being ready for training:

- **Filtering:** Generalizing the data from futile or private information, replacing all dates, hours, emails and generic numbers with more general tokens, namely <DATE>, <HOUR>, <EMAIL>, <NUMBER>. Particular care was put into the email processing, a process which was manually checked to ensure the avoidance of personal data in the final training dataset, which might have happened due to the possible incorrectness of human written text.

- **Tokenization:** Splitting the data in single words and removing punctuation. A particularity here stands in the need for a fixed number of words in the processed *Oggetto* field. The use of a Transformer (further introduced in **Section 3.1.1**) implies in our case an Embedding Layer for which a fixed input dimension is required. We therefore studied and analyzed the *Oggetto* length variability and opted for a sequence length of **30 words**. The decision came from a statistical analysis (a summary can be appreciated in **Figure 2**) merged with the will of being as indulgent as we can and maintain as much information as possible in the sentence. Consequently, sentences with more than 30 words are truncated and in the opposite case are filled to reach the correct length with special padding tokens (<PAD>).

- **Encoding:** After collecting all dataset tokens a vocabulary is created and so is a mapping between token and integer. This allows for a fast encoding procedure, transforming the input sentence in an integer array which can be directly fed to the model (if a word is not present in the vocabulary it is replaced with a special token, <UNK>).

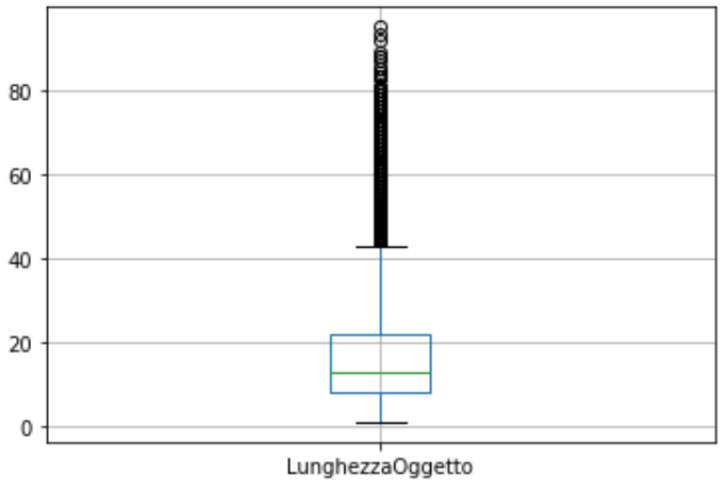


Figure 2: Object length boxplot

3 First Level Model

In this section is described the model used to perform the classification in the first (highest, most general) level of the hierarchy of classes.

3.1 Model Description

The class of each protocol depends on two aspects:

- **Oggetto**: subject of the document, inserted by the human operator at classification time.
- **Contenitore**: archiving office for the protocol. Each office may archive different categories of documents, which, therefore, belong to different classes.

These two attributes are treated differently, by two different sub-models (**object module** 3.1.1 and **office module** 3.1.2) whose outputs will be combined and fed to a multi-layer perceptron (MLP) that will perform the final classification.

3.1.1 Object module

Due to the textual nature of *Oggetto* attribute, that can be seen as a summary of the content of the document, it is important to capture its semantics in order to understand the general matter of the document itself and perform a meaningful classification.
For this reason, the choice was to start from a standard **Transformer** [1, 4], that is formed by a stack of encoders and decoders that work as follows:

- **Encoders:** each one of the encoders of the stack has two sub-layers: the first is a *multi-head self-attention* mechanism, and the second is a simple, position-wise fully connected feed-forward network.
- **Decoders:** each decoder in the stack, in addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack.
- **Multi-head self-attention:** Instead of performing a single attention function, *queries*, *keys* and *values* are linearly projected h times with

different, learnt, linear projections. The attention function is performed in parallel for each of these queries, keys and values.

Now, since the task is a classification (and not, for example, a sequence-to-sequence transduction), only the encoders of the transformer are needed.
The model used in the task, so, is formed by an embedding layer [3] followed by a *TransformerEncoder* [5], that is a stack of encoders as described in [1].

3.1.2 Office module

The information about the office that performs the classification is categorical, so, from textual data, it gets 1-hot encoded. This, anyway, is not beneficial for the model, because the information would be very sparse, for this reasons **office module** serves as an embedder used to move towards a dense representation.

Architecture This embedder is built as a MLP with 3 layers, all with a ReLU activation function. In particular:

- input layer: 20 units, i.e. the dimensionality of the 1-hot encoding of the office.
- hidden layer: 10 units, it reduces the dimensionality to serve as an "information bottleneck".
- output layer: 128 units, i.e. the chosen dimensionality of the embedding.

Training of the embedder This embedder is trained in an *encoder-decoder fashion*: at training time, another fully-connected layer is added at the end of the embedder, whose input dimension corresponds to the embedding one (128) and its output dimension is equal to the dimensionality of the original 1-hot encoding of the office (20). This final layer has a *tanh* activation function and its purpose is to reconstruct the original 1-hot encoding. To draw a parallel with an encoder-decoder architecture, the embedder acts as the encoder while the final layer serves as decoder.
During the training process, the data is passed through the full architecture (embedder + final layer) and the loss function (cross entropy) forces the model's output to be as similar as possible to the model's input.

Notes:

- PyTorch's implementation of the cross entropy loss [2] does not necessarily want an input that is scaled in the interval $[0, 1]$, as this operation is performed by the loss itself.
- The *sigmoid* activation function in the final layer would also be good for reconstructing a 1-hot encoding, but the results were worse, therefore the final choice was the *tanh*. As stated in the previous point, it is not necessary that the model's output is scaled in $[0, 1]$.

3.2 Performances

Experiments were carried out on a *Lenovo Legion Y740-17IRHg* using its GPU (*NVIDIA GeForce RTX 2080 Max-Q*) to speedup the training process. Pay attention that although trainings are executed on GPU, all timings reported for inference are referring to CPU execution of a trained model.

Model training and validation have been carried out using a dataset with data coming from 2018 up to mid May 2021 using a *Hold Out* validation strategy with an 80-20 split. *Top 1* and *Top 3* accuracy are used for model selection and *Cross Entropy Loss* is used for learning because of the multi-class task. *ReLU* is used throughout all the model as activation function.

An initial screening phase brought out the necessity of having a bigger chunk of the final classifier input composed of the object module (**Section 3.1.1**) with respect to the office module (**Section 3.1.2**) embedding size. The screening phase also helped with fixing other parameters, such as the *Learning Rate* (after the initial trials we opted for a default *Adam* optimizer), the use of a *Multi-Head Attention* with 8 heads and a Transformer Encoder stack of 2 levels.

Our trials have therefore focused on attempting the varying of *object embedding size*, *office embedding size*, *number and extension of layers in final classifier* and *amount of regularization*. Most relevant results are shown in **Table 1**. All attempts[1] are given 50 epochs but have stopped before reaching the end because of a *patience* mechanism.

[1] obj: *object module*, off: *office module*, emb: *embedding*, cls: *final classifier*

Model	Off/Obj Emb Size	Obj Output Size	Cls Topology	Obj/Cls Dropout
model 25	128/256	256	(obj + off, 15)	0.777/0.777
model 35	128/256	256	(obj + off, 512, 256, 15)	0.5/0.5
model 36	128/256	256	(obj + off, 256, 15)	0.5/0.5
model 38	128/256	256	(obj + off, 15)	0.5/0.0
model 39	128/512	512	(obj + off, 15)	0.5/0.5
model 40	128/128	128	(obj + off, 15)	0.5/0.5

Table 1: *Hyperparameters* of most relevant trials

Model	Top1 Tr/Val	Top3 Tr/Val
model 25	95.70/87.82	99.48/95.70
model 35	91.27/86.79	97.18/95.16
model 36	89.67/86.39	96.52/94.96
model 38	90.41/86.21	97.40/94.77
model 39	92.55/87.78	98.62/95.68
model 40	91.80/86.91	98.21/95.50

Table 2: *Performances* of most relevant trials

Results shown in **Table 1** and **Table 2** give rise to some interesting observations. It can be noticed in fact that the topology increment of the final classifier does not help the model to learn better and neither does increasing the object module output size (even if the worsening in accuracy is minimal). In the screening phase we also noticed a similar minimal phenomenon in worsening of accuracy when increasing the number of transformer encoders stacked onto each other.

The selected model for the first level task is eventually **Model 25**, for which, taking a look at the model training/validation plot (**Figure 5**), overfitting is clear even if the model's regularization is remarkable. This event happened in all models we attempted, with the only exception being the *office only model* (**Section 5.1.2**) which underperformed nevertheless.

The event of overfitting can be explained in this application case by the dataset nature. The dataset is highly unbalanced and needed balancing as explained in **Section 2.3**. The balancing process was a special kind of stratification, which lead to the presence of many extremely similar (or in some cases directly replicated) samples. Still, the presence of overfitting does not compromise the generalization capability of the model. In fact, it can be appreciated how the validation loss keeps lowering as the model continues its training until we reach the real moment of overfitting (highlighted by the red *patience* bar), beginning of the *patience* mechanism.

4 Second Level Model

Here is described the model used to perform a classification that reaches the second level of the classes' hierarchy.
This task is more challenging because the number of classes grows significantly - from *15* to *118* - and the data available gets more sparse and unbalanced (also because of the increasing level of detail).

4.1 Model Description

The model is composed of 3 main modules:

- **object module:** same as for the first level model (**3.1.1**).
- **office module:** same as for the first level model (**3.1.2**). It is loaded from a pretrained version used for the first-level classification and its parameters are kept frozen[2].
- **first level module:** consists of the pre-trained first level model (**Sec. 3**). It is loaded and its parameters are kept frozen[2].
- **final classifier:** a final MLP to perform the actual classification.

The input-output process works as follows:

1. A first-level classification is performed using the *first level module*
2. An embedding of the office is obtained through the *office module*
3. The object is passed through the *object module*, the result is flattened and then sent to a FC layer[3] with ReLU activation function to get the desired dimensionality
4. The outputs of the steps 2 and 3 are concatenated, some dropout is applied and the result is sent to the *final classifier*
5. Finally a bias is added to the output of the *final classifier*. The output in this case is a vector of 118 cells, each cell represent a second level class, which is a refinement of a first level one. Therefore, exploiting

[2] "Frozen" parameters (weights) are immutable/non-trainable.
[3] fully-connected / dense.

a mapping between *first-level class* and *second level output index*, it is possible to add a bias accordingly to the first level prediction. In this way, the second level classes that corresponds to a more likely first level class will receive a boost.

Exploiting the results of the first level prediction in order to guide the second level may be dangerous if the first level is not precise. In our case given the results presented in **3.2** the first level module has a 99% *Top 3* accuracy, so the bias injected in the second level prediction in most cases does not provide any noise. **Section 5.2** analyzes the results obtained without using the first-level module output information.

4.2 Performances

As expected, the harder the task (in this case we enter in a situation where many more classes are present and data is a lot sparser) the lower the accuracy of the model. After a second screening phase we maintained the validation method and the same fixed parameters of the first level task (see **Section 3.2**). Because of the poorer results we decided to introduce a *Top 5* accuracy. The introduction of a broader new accuracy assures a good result while still keeping an objectively restricted choice for the client.

Model	Off/Obj Emb Size	Obj Output Size	Cls Topology	Obj/Cls Dropout
model 41	128/256	256	(obj + off, 15)	0.5/0.2
model 42	128/512	512	(obj + off, 15)	0.5/0.2
model 43	128/128	128	(obj + off, 15)	0.5/0.2
model 45	128/256	256	(obj + off, 15)	0.7/0.7
model 46	128/256	256	(obj + off, 512, 256, 15)	0.5/0.2

Table 3: *Hyperparameters* of
second level task most relevant trials

Model	Top1 Tr/Val	Top3 Tr/Val	Top5 Tr/Val
model 41	84.67/78.35	94.52/87.20	97.35/90.53
model 42	67.88/65.67	86.83/82.12	92.89/87.98
model 43	84.14/77.71	94.61/86.95	97.15/90.30
model 45	67.26/65.55	86.96/82.17	92.90/87.76
model 46	83.95/76.93	94.10/86.26	96.98/90.10

Table 4: *Performances* of
second level task most relevant trials

Most relevant results are shown in **Table 3** and **Table 4**. All attempts are given once again 50 epochs but have stopped before reaching the end because of a *patience* mechanism.

As can be appreciated, even if the *Top 1* accuracy is not astonishing, **Model 41** reached a stunning 90% in *Top 5* validation accuracy, making it the selected model for the second level task. In particular, making a parallel with the first level performance analysis, the same phenomenon of overfitting can be appreciated in the model's training and validation plots (**Figure 6**), even more accentuated because of the sparsity of the task.

5 Relevant Attempts

Here is provided a list of relevant alternative attempts made during the development of the project.
They can provide interesting comparisons and insights about the behaviour and characteristics of the main models presented in **Sections 3** and **4**, as well as a different point of view on the *Contenitore* field (e.g. **Section 5.1.3**).
Some models were developed for a first-level classification (**5.1**) and an alternative attempt has been made for a 2-level classification (**5.2**).

5.1 First level

Here are presented alternative models used and tested on the first level of the classes' hierarchy.

The models used in Sections **3** and **4** exploit information coming from two sources:

- object: the real textual information, it is a summary of the content of the document and it is important to capture its meaning in order to correctly classify the document;
- office: it is more of a categorical information providing hints on the subset of classes that is more likely to contain the correct one.

It is possible to operate a trade-off between the contributions of these two pieces of information by changing the relative dimensions of the outputs of

the *object module* and the *office module* [4].
This concept, if brought to the extreme, leads to two different and opposite situations: one where only the object is taken into account (Sec. **5.1.1**) and the other where only the office is considered (Sec. **5.1.2**).

5.1.1 Object only

This model tries to classify the documents considering only its object, with no information about the office that manages this protocol. This test is interesting for understanding exactly how well the model can adapt to this situation where a part of the information is missing and how important is one field compared to the other[5] for the final outcome.

Model description This model simply removes the *office module* (**3.1.2**): the object gets fed into the *object module* (**3.1.1**) whose output is flattened and sent to a FC layer[3] to get the desired dimensionality (it is reduced). Finally some dropout is added and the data is sent to the final classifier, which, in this case, is a single FC layer[3].

Performances Given the underlying structure of this model - **3.1.1** - nice results can be achieved even without the office information. This is given by the fact that the most characterizing information for the classification task is the object. While the same office can handle different classes, the same object is always classified in the same way and rarely changing few elements in the text change the classification. Nevertheless, its performances are worse than the model presented in **Section 3** as reported in the following table.

Model	Top1 Tr/Val	Top3 Tr/Val
Model 25	95.70/87.82	99.48/95.70
Obj only	87.42/84.13	96.41/94.30

Table 5: Comparison between *first-level* and *object only* models

[4]The output of the *object module* (o_1) and the output of the *office module* (o_2) are concatenated to form the input of the final classifier, therefore, if o_1 is much larger than o_2 (or vice versa), it is going to represent a bigger portion of the classifier's input and it will be more relevant for the final outcome.

[5]The fields are *Oggetto* (object) and *Contenitore* (office).

5.1.2 Office only

In this case, the model tries to classify the documents using only the office information, ignoring the object field. As highlighted in the previous subsection, this analysis helps to understand how the single information performs alone and why merging the two provide better results.

Model description This model is obtained by removing the *object module* (**3.1.1**). The *pre-trained office module* provides an encoding of the office data, then this high dimensional representation goes through a FC Layer[3] to get the correct output size. The model is then trained to solve the document classification task.

Performances In this case the results are definitely worse with respect to the *first-level* or *object only* model. This is related to the limited size of the model and its parameters, only the last layers are in fact updated, but also to the office information. As mentioned in the previous case, one office can handle documents that are classified differently, leading to pairs of input/output that are not always the same. This two aspects strongly influence the performances of the model, even with different sizes of encoding. Results are reported in the following table.

Model	**Top1 Tr/Val**	**Top3 Tr/Val**
Model 25	95.70/87.82	99.48/95.70
Obj only	87.42/84.13	96.41/94.30
Off only 128	58.48/58.79	78.92/80.14
Off only 256	58.99/58.86	80.02/80.53
Off only 512	58.78/58.78	80.03/80.60

Table 6: Comparison between *first-level*, *object only* and *office only* models

5.1.3 Office's classes distribution as injected prior probability

As stated in **Sec. 3.1.2**, the office where a certain document is archived is a **categorical** information. Previously (Sections **3** and **4**), this information has been moved from a 1-hot representation, where all the offices are equidistant from one another, to a dense representation exploiting an embedding

mechanism. This because different offices may actually share similarities, therefore a dense representation is more effective for the classification task at hand. These similarities are not something abstract that can indeed be captured by the embedding but not interpreted by humans; in this specific case, similarities among offices are simply reflected in similarities among the *distributions of the classes associated to each office* (referred to as **classes distribution** of an office (Def. 1)).

Definition 1 (Classes distribution) *The **classes distribution** of an office is a vector as long as the number of classes where each entry represents how many documents archived in that office belong to each class, expressed in percentage* [6].

Consider this example:

- there are 15 classes.
- the entries archived in office A belong for 70% to class 1 and 30% to class 3.
- the entries archived in office B belong for 80% to class 10 and 20% to class 11.
- the entries archived in office C belong for 80% to class 10, 10% to class 11 and 10% to class 13.

It is easy to conclude that office B and office C are more similar to each other than any of them is similar to office A, in the sense that they share some areas of competence and they manage similar kinds of documents. As a consequence, their classes distribution vectors will be more similar to each other than any of them is to the classes distribution of office A.
This concept is exploited in the following alternative model.

Model description The *object module* stays the same, but instead of concatenating its output with the *office module*'s output and then sending the whole to a final MLP, the information regarding the office is injected into the

[6] Therefore, each office will have a percentage/probability distribution over the classes. For example, if all the documents archived in *office A* belong to class 0, its **classes distribution** will be a vector as long as the number of classes that will look like this: [1.0, 0.0, ..., 0.0].

model as a **prior probability**.
Practically this is done in the following way:

- For each input:
  - Check the office (*Contenitore* field);
  - Read the classes distribution vector of that office;
  - Perform an element-wise multiplication between the output of the final MLP and this classes distribution vector, scaling the output accordingly to the distribution of the classes.

This will have the effect of adding a bias to the output of the final MLP (that does not consider the office) taking into account that the office of the current entry manages only certain types of documents associated to certain classes in a certain measure.

Performances Given the architecture of this model, that exploits the *object module* and combines it with prior probabilities, the results were expected to be more or less similar to the object only scenario (**5.1.1**). This expectation was in fact confirmed: the models performances are very close. The only difference between the two are the prior probabilities. These can affect model prediction by "boosting" some classes exploiting the distribution of the office. Since the office information is not the perfect oracle as analyzed in **5.1.2** this introduce some noise to the model leading to slightly worse performances. Theoretically reducing the influence of the prior probability to zero the model performances will asymptotically reach the one of **5.1.1**. The following table reports the results and a comparison with the object only model.

Model	**Top1 Tr/Val**	**Top3 Tr/Val**
Model 25	95.70/87.82	99.48/95.70
Obj only	87.42/84.13	96.41/94.30
Obj + priors	84.67/81.23	93.25/91.23

Table 7: Comparison between *first-level*, *object only* and *offices as priors* models

5.2 Second level

Section 4 shows the "standard" model to perform a classification up to the second level of the classes' hierarchy.
The next subsection instead describes an alternative attempt made without using the first-level classification to refine the second-level one.

5.2.1 Without first level prediction bias

This model can be seen as a variant of the one described in **Section 4**, but the prediction of the second level is *completely independent* from the first-level class (actual or predicted) of the document in question. In this sense, the concept of this model is much more similar to the one of **Section 3** (i.e. first-level classification).

Model description Here, the *first level module* is not present, making the *second level module* totally unbiased.

The second-level classification is performed as in the beginning of the standard *second-level model* (**4**): the office and the object pass through their respective modules, then the latter is flattened and its dimensionality reduced (as always[7], using an apposite FC layer[3]). Finally these are concatenated and, after applying some dropout, they are sent to the final classifier (i.e. a MLP).

Performances Unlike other models, the performances are in this case surprisingly slightly better than the second level task chosen model, requiring a deeper analysis to understand where and how this simpler model outperforms model 41 (**Section 4.2**). Taking a look at **Table 8** the model is able to snatch almost a full *Top 1* validation accuracy point with respect to the chosen model.

Model	**Top1 Tr/Val**	**Top3 Tr/Val**	**Top5 Tr/Val**
Model 41	84.67/78.35	94.52/87.20	97.35/90.53
Model No Bias	87.06/79.32	95.72/87.91	97.80/91.14

Table 8: Comparison between *second-level* and *second-level no bias* models

[7] "always" refers to the models described in sections 3, 4, 5.1.1 and 5.1.3

For a better insight we increased our level of analysis to the single class accuracy. The objective was to understand in which classes the two models perform the best given two quantitative measures:

- **First class accuracy**: a measure (per class) of how accurate is the second level model in classifying a second level class at least in the correct first level.
- **Second class accuracy**: a measure (per class) of how accurate is the second level model in classifying a second level class in full precision.

Unfortunately this more in depth analysis did not give better insights. This is probably due to the incredibly near performances on the validation dataset and will be more in depth studied on the test dataset (**Section 6**).

6 Tests

CTT company kindly provided us with a set of data completely detached from the previous dataset. This **test set** is in fact composed of the documents following the previous period, in particular going from mid May 2021 to mid July 2021. The presence of such data allowed us the access to a true **model assessment** phase for our final models.

6.1 First Level

Taking a look at the easier task, **Table 9** represents the accuracy comparison between validation and test set. The results obtained from **model 25** are quite good, reaching *82% Top 1* accuracy and *93% Top 3*. This further prove, that the training process provided a good generalization capability to the model which leads to nice results over a new set of data.

Phase	**Top1**	**Top3**
Validation	87.82	95.70
Test	82.08	93.18

Table 9: Comparison between Model 25 *validation* and *test* performances

Figure 7 displays the *confusion matrix* over the test set exploiting the *Top 1 prediction* of the model. First thing that stands out is the distribution of the samples which is very unbalanced, similarly to the original dataset, up to the point that *class 5* has 0 samples. As far as concerns other classes, the low represented classes - *3, 11 and 12* - are the ones where the misclassification is higher. Other classes suffers the same trend of error, and this can be justified by errors made from the users - either on the class or the office - but also from some changes of competence between offices over documents during time. The latter reason also emerged during a meeting with Doc. Franchi, where a small live demo was set up. During the test, the *Top 1* prediction of the model was incorrect using a known object and one possible document repository, but simply changing the latter was enough to get the correct prediction with *99%* reliability. This shows that the model has a nice generalization capability which, however, is very susceptible to the input data and this is obviously due to the data used to train the model. Nevertheless, even in the first test case, considering the *Top 3* prediction was enough to have the correct answer with *30%* reliability. The better accuracy of the model using the *Top 3 prediction* can be seen in **Figure 8**. The confusion matrix highlights that all classes, except class 12, have more than 50% accuracy with few misclassified samples only in the low represented classes. **Figure 9** shows for each class the prediction accuracy comparing the *Top 1* and *Top 3* prediction, suggesting that providing the latter to the user would for sure provide a trustworthy suggestion.

6.2 Second Level

Moving to the more complex task, **Table 10** reports the results obtained in the second level classification task. Also in this case, the accuracy of the model is similar to the one obtained during the process of validation.

Phase	**Top1**	**Top3**	**Top5**
Validation	78.35	87.20	90.53
Test	72.49	82.58	85.82

Table 10: Comparison between Model 41 *validation* and *test* performances

In this scenario providing a confusion matrix or representation of the accuracy over the *118 classes* would not have been the optimal choice, so the focus of the analysis was shifted to more general statistics. In particular the *Top 1* and *Top 5* prediction will be taken into consideration and compared to analyze the performances of the model. **Figure 11** displays two pie plots: **11(a)** reports *Top 1* prediction statistics, while **11(b)** depicts *Top 5* prediction statistic.
Starting from the first one, the bigger portion - **38.3%** - refers to the percentage of classes with *0% accuracy*, this is related to a huge number of classes that only have few samples in the test set as well as in the training set. This low represented data might not be correctly generalized by the model and lead to extremely poor accuracy. Nonetheless, **almost 50%** of classes have *at least 25% accuracy*, and **more than 25%** have *at least 50% accuracy*; results that - considering a *Top 1 prediction* - are not bad at all.
Considering the second plot the performances improve significantly. The number of classes with *0% accuracy* is lowered to **25%**. The bigger portion - **34.6%** - in this case refers to classes with over *75% accuracy*, and **almost 60%** of classes have *at least 50% accuracy*. Similar to the previous case, the *Top 5 prediction* provides the user a viable suggestion from which to choose the class for the document.

6.3 Inference Time

A key and fundamental point that goes beyond the accuracy numbers and affect the model performances is the inference time. In fact the proposed solution should not only be precise, but should be able to provide the user a set of possible classes in a small interval of time. As presented in **Section 1** and reported in **Figure 4**, currently the user has to click and navigate through the various levels of the hierarchy in order to find the correct class. If the model is too slow the task is easily solved by the user, on the other hand if the model rapidly provides an answer, it can be exploited to avoid searching.

Model	**Inference time (*sec*)**
Model 25	0.002777
Model 41	0.006295

Table 11: Inference time *first* and *second level* models

In order to get a fair estimate, the model was loaded on the *CPU*, in this way it would be tested on a normal hardware available in any device. The time reported in **Table 11** are an average of 10 different tests.

Both architectures have a very small inference time, order of milliseconds, resulting in a very efficient solution. The users will be able to select the office, write the object and almost instantly get the result. If the model, in the 3/5 possible classes, provides the correct one the user will be able to select it without using the class hierarchy interface.

6.4 Second level with/without bias comparison

The unclear insights in analyzing the better performances of **Section 5.2.1** become clearer on a different dataset as the test set. The performances can be appreciated in **Table 12** where *Model No Bias* achieves a result with more than a full accuracy point over *Model 41*.

Model	**Top1**	**Top3**	**Top5**
Model 41	72.49	82.58	85.82
Model No Bias	73.11	84.08	87.03

Table 12: Comparison between Model 41 and Model No Bias *test* performances

The deeper analysis already attempted in **5.2.1** is reproduced in this case and the results are shown in **Figure 10**. Such a behaviour shows a supremacy of the Model No Bias in less populated classes (such as 9, 10, 12), demonstrating how an uncertain or erroneous first level bias leads to a worsened result with respect to a simpler and unbiased model. On the other end, highly populated classes (or anyway classes where Model 41 has a more decise and clean result) tend to perform better on a biased model such as Model 41. Therefore the **tradeoff** is clear and paves the way to a decision driven by the needs of the company. Both the models are definitely performing well and both could be used, but this experiment was nevertheless important to understand that a finer grained analysis can give rise to better model explainability.

7 Model Maintainability

The purpose of this project is to provide a viable solution that can help the users in the classification task. This solution should adapt and evolve with possible changes that can happen inside the company, e.g. new offices, changes of competence etc. As time goes by, more and more data is handled and classified by the users which leads to new sets of entries that can be exploited to improve model's performances.

There are different approaches that can be used to update the model. It can be retrained from scratch using a bigger dataset, or it is possible to keep the current model. While the first alternative is more brutal, since we re-train the previous model, the second one keeps the current one and tries to improve it. This approach can be seen as a **fine tuning**, taking inspiration from the homonymous process that can be used for models such as pre-trained BERT.

In particular, trying to apply this idea to the proposed solution, different strategies have been analyzed:

- Apply fine tuning to *first level model*.
- Apply fine tuning to *second level model*, keeping the same first level module.
- Apply fine tuning to *second level model*, updating the first level module.

The models that have been analyzed in the previous sections are trained using the dataset presented in **Section 2**. The data was split into training and validation set. In this part of the analysis, the validation set is used as training set for the fine tuning with **7988** entries. Despite the small size of the training data and few epochs of training, the results lead to an improvement in the performances in the second level classification task over the test data (ref. **6**).

Here is reported the accuracy comparison over the test set considering the following models:

A. Second level model (ref. **4**).

B. Fine tuning of model **A**.

C. Fine tuning of *first level module* (**4.1**) of model **A**, then fine tuning of the resulting *second level model*.

Model	Loss	% Top1	% Top3	% Top5
A	1.82532	72.50	82.58	85.83
B	1.30638	73.05	83.56	87.15
C	1.46075	73.52	83.56	87.44

Table 13: Comparison over test data of second level models

8 Conclusions and Future Development

In this report different solutions to solve Document Classification in public administration have been analyzed. The different experiments for the *first and second level* classification task lead to interesting results. In the first case, the easier one with only 15 classes, the model is able to reach high performances - *93% Top 3 accuracy* - and also provide nice results for all the classes, going beyond their unbalanced distribution. The second one instead, was the most challenging task given the significantly higher number of classes and a much sparser distribution. Nevertheless, different solutions provided interesting results that went beyond the expectations, reaching *85% accuracy* exploiting a Top 5 prediction. The proposed solutions proved to be a viable integration to the current process of document classification providing both speed and precision. *CTT* during the whole period showed to be excited about the project and surprised by the results achieved. We would like to thank them for the availability and support.

Last but not least, we will keep working on the project creating a more general solution. The idea is to provide a service that can be used by any company that, with the correctly tuned model, will be able to exploit its prediction to improve the process of classification.

9 References

[1] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[2] *PyTorch's cross entropy loss documentation*. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Accessed: 2021.

[3] *PyTorch's embedding layer documentation*. https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html. Accessed: 2021.

[4] *PyTorch's transformer documentation*. https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html. Accessed: 2021.

[5] *PyTorch's transformer's encoder documentation*. https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html. Accessed: 2021.

10 Figures

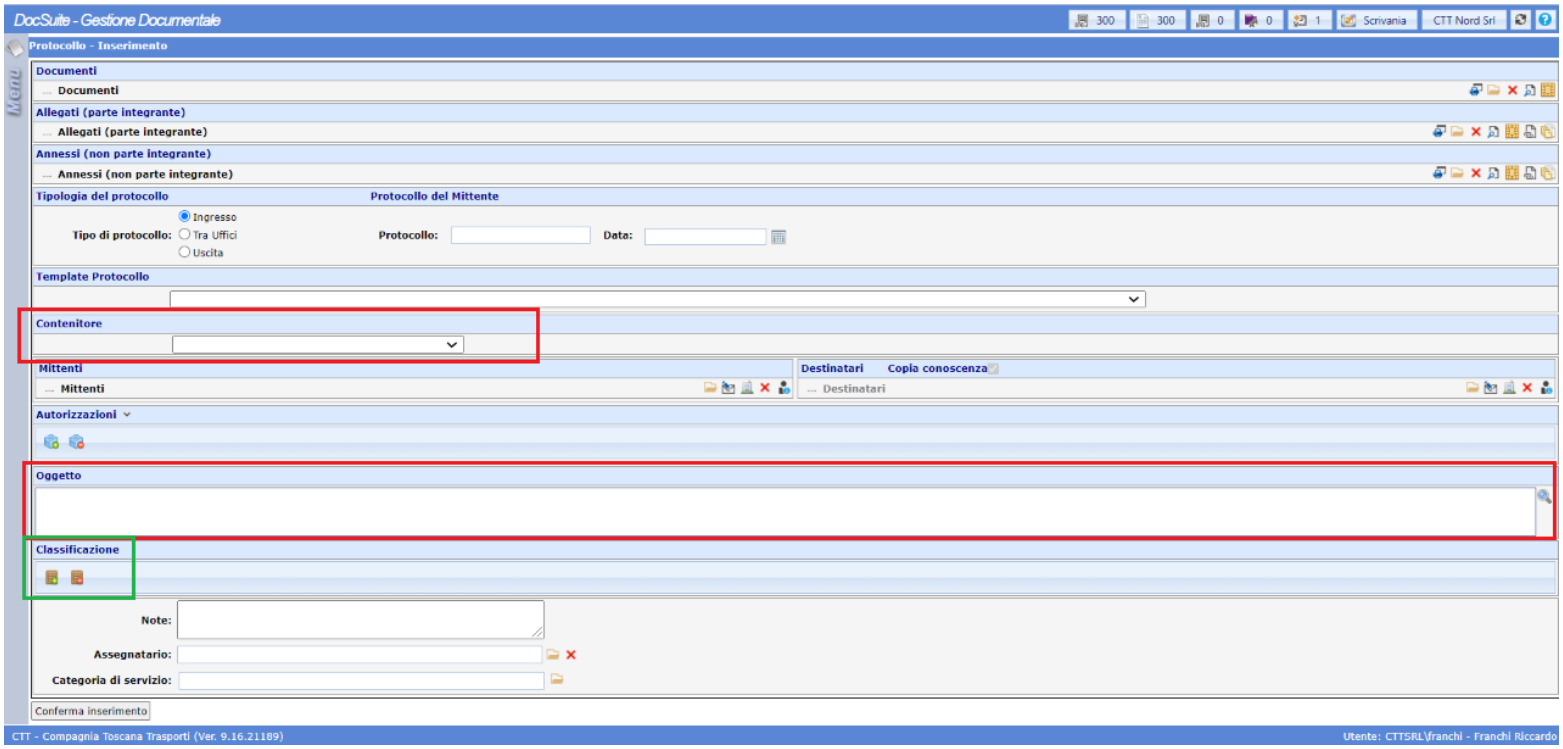


Figure 3: Public administration user interface

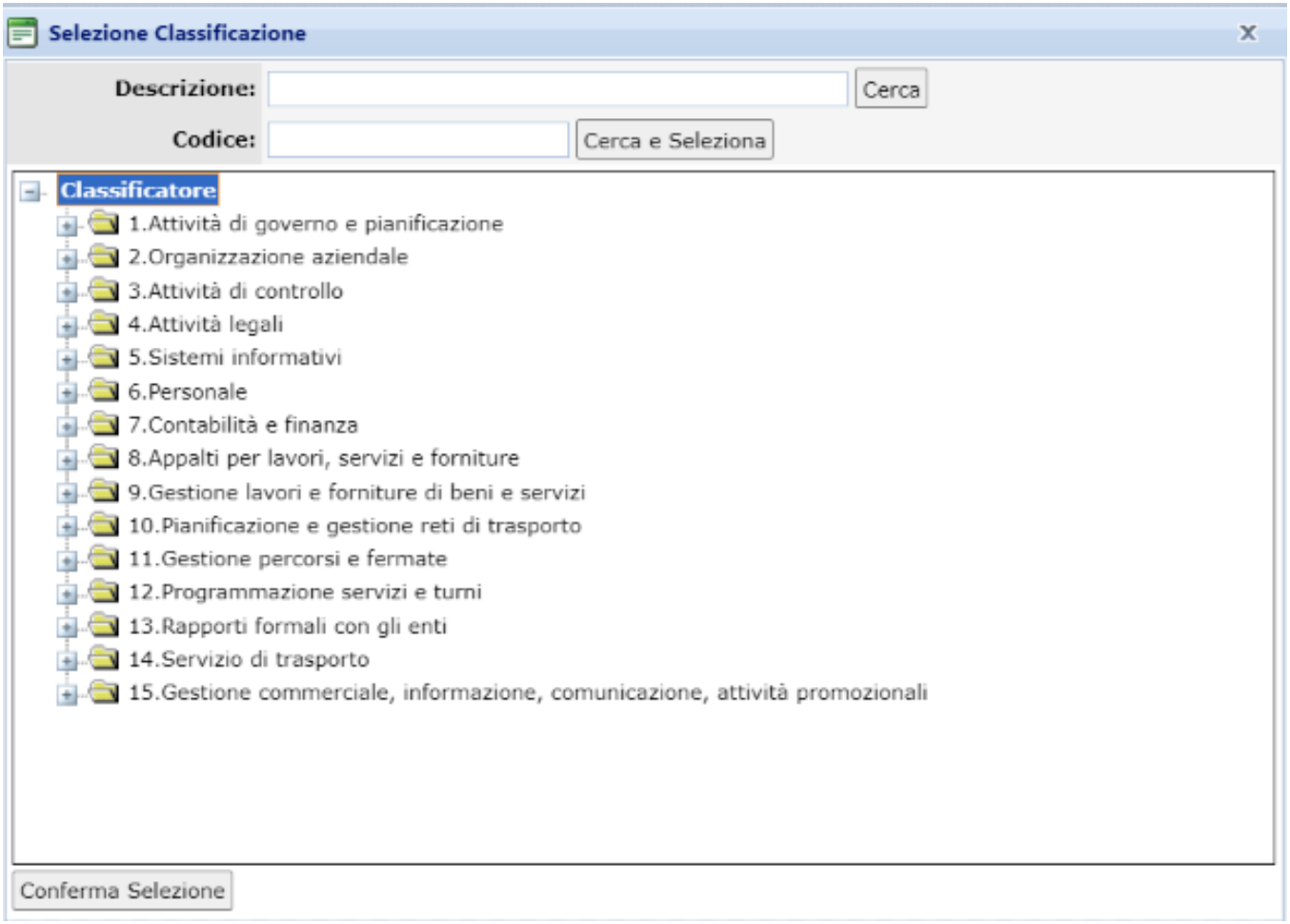


Figure 4: Classes hierarchy user interface

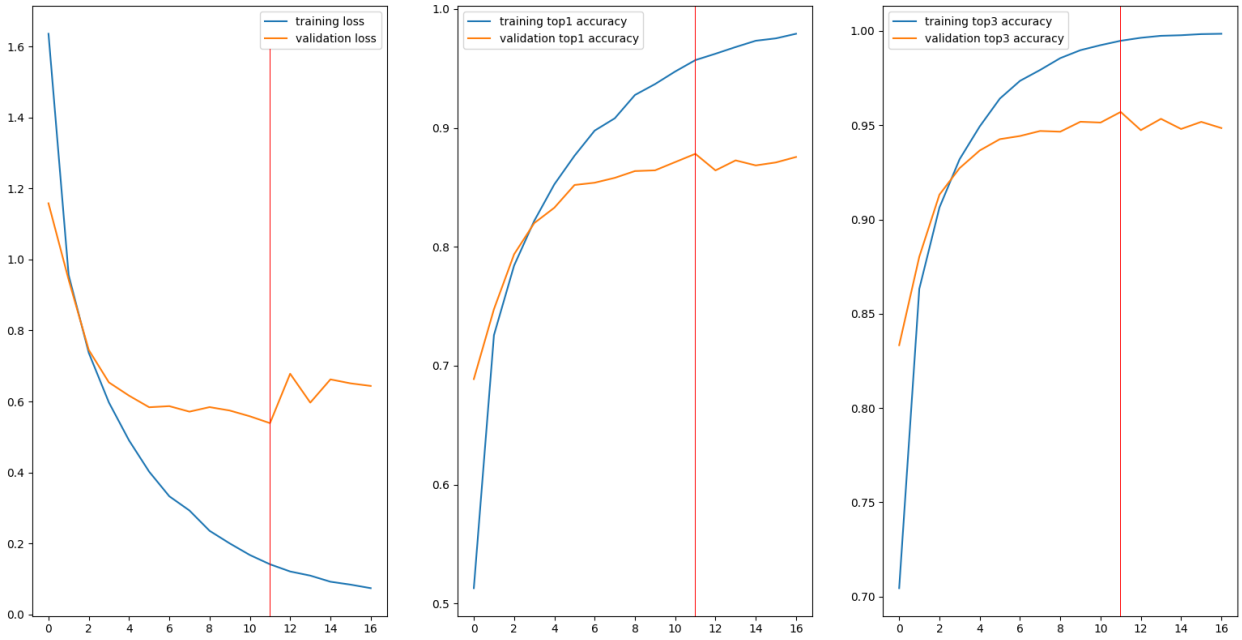


Figure 5: Model 25 training/validation plots

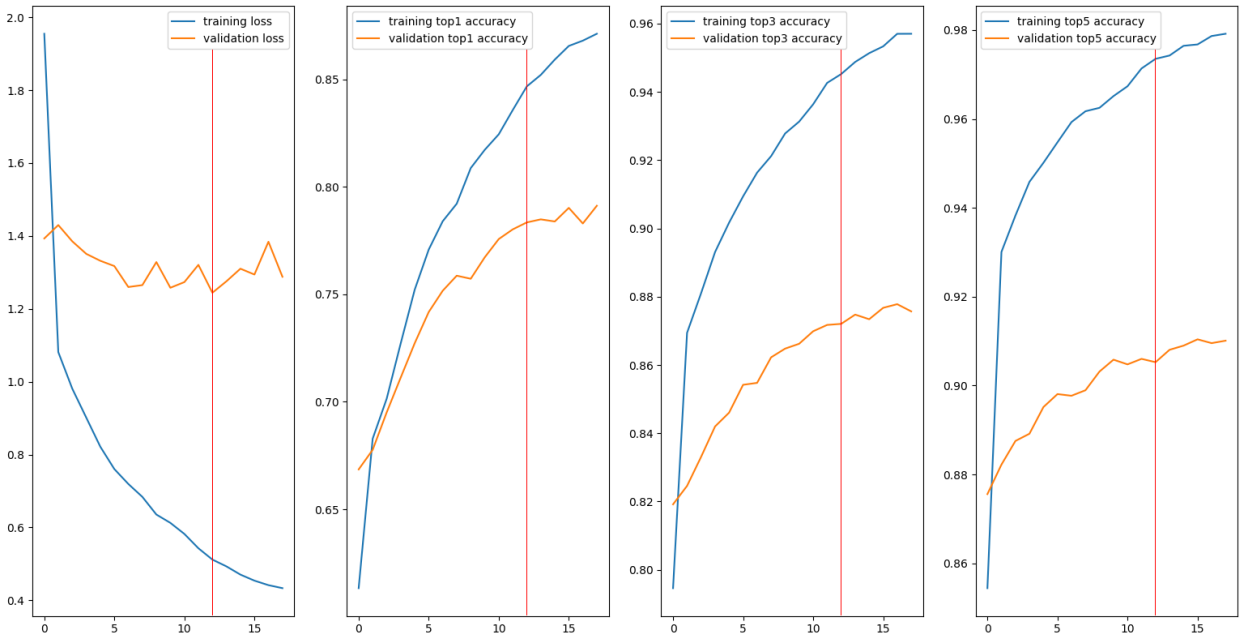


Figure 6: Model 41 training/validation plots

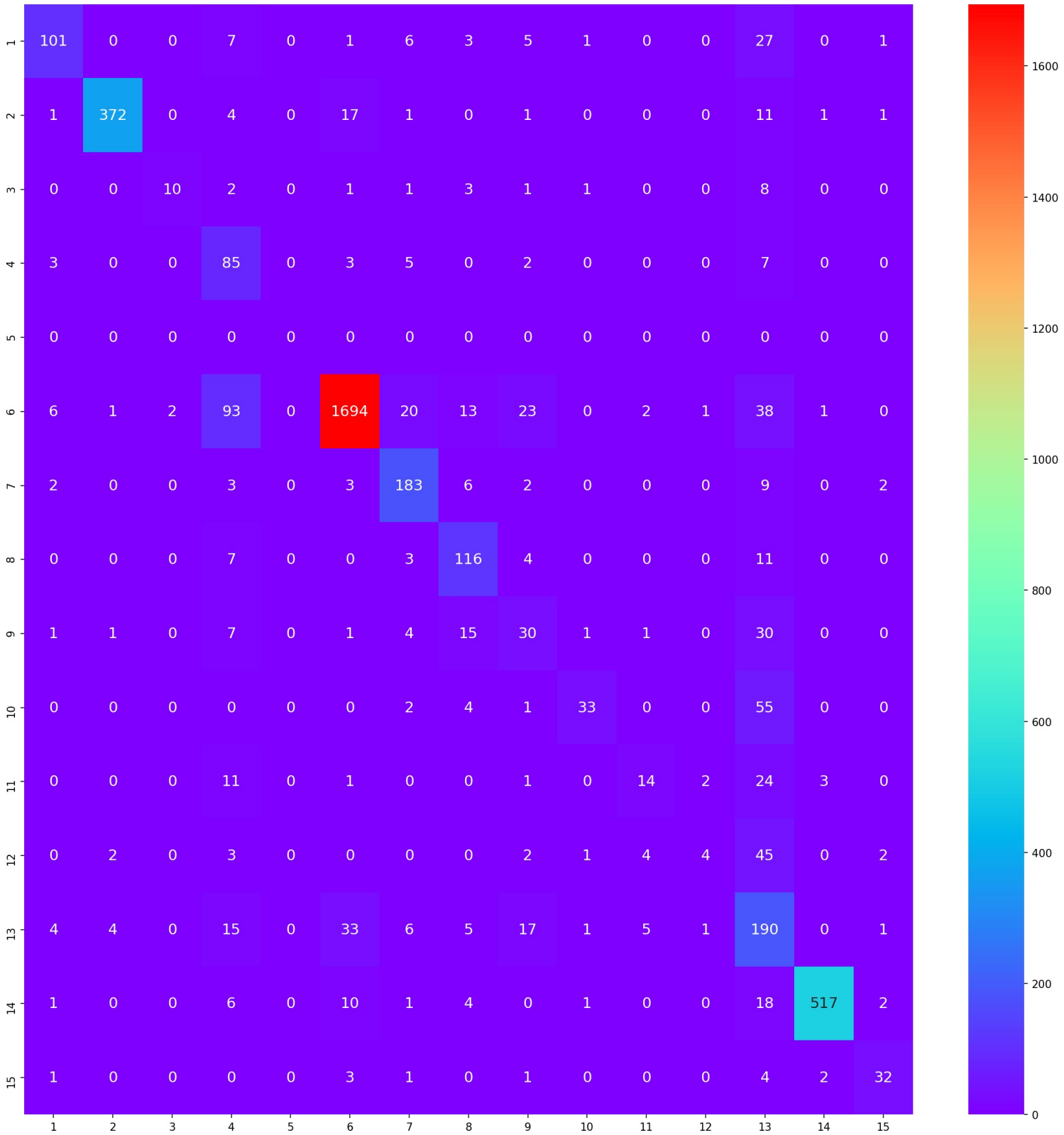


Figure 7: Heatmap for Test Set on First level task (*Top 1*)
y axis: *ground truth* - x axis: *model prediction*

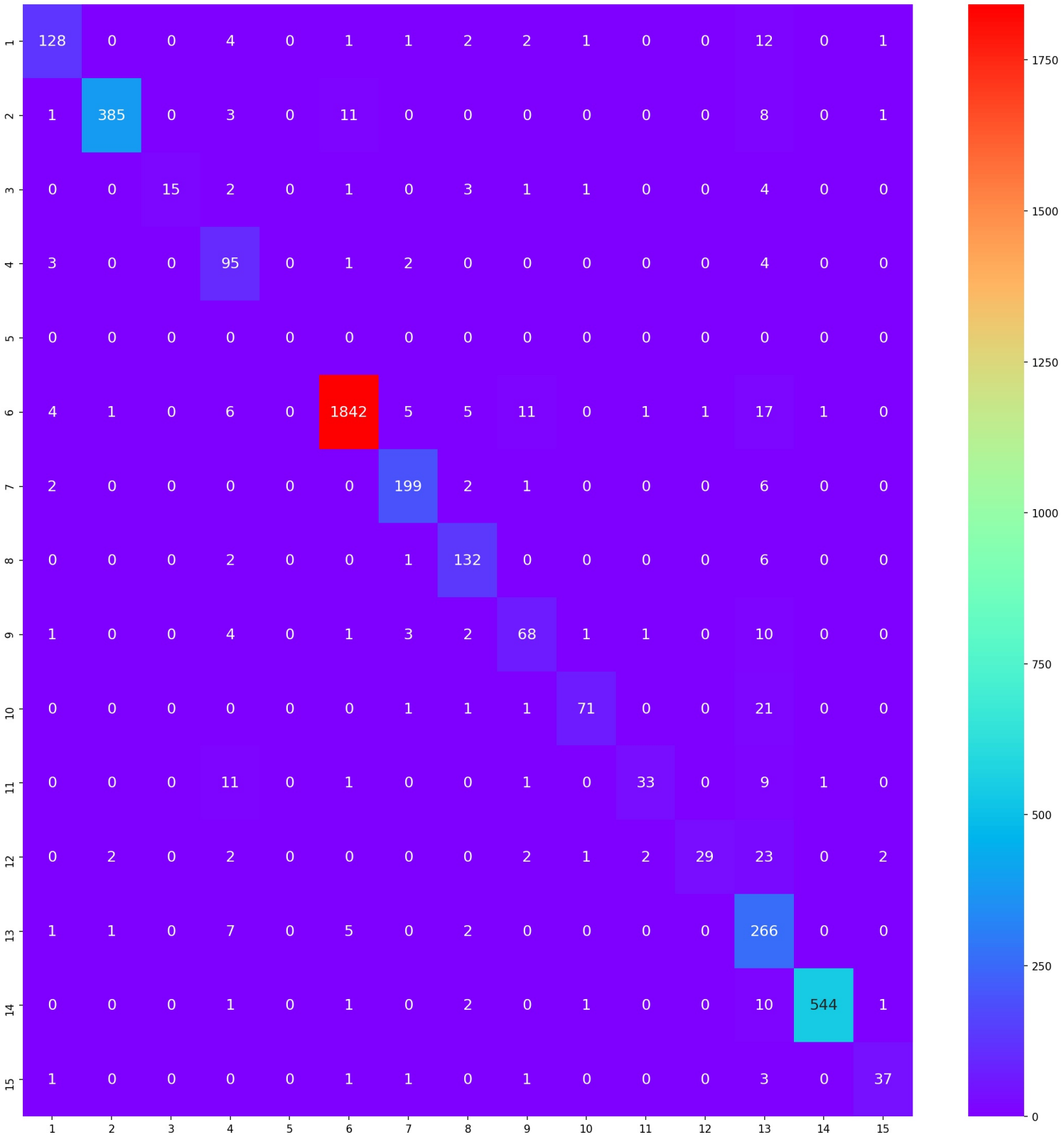
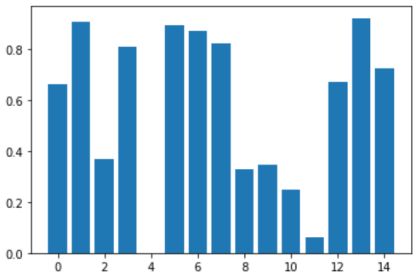
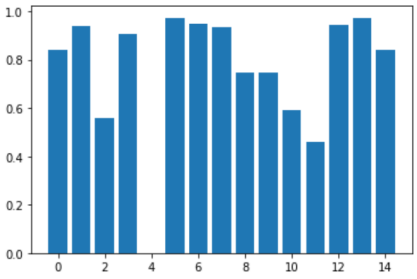


Figure 8: Heatmap for Test Set on First level task (*Top 3)*
y axis: ground truth - x axis: model prediction

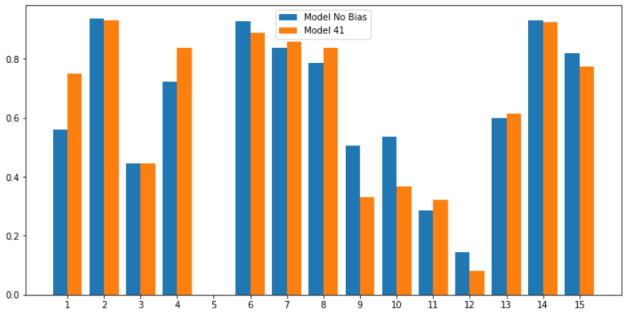


(a) Top 1 Accuracy

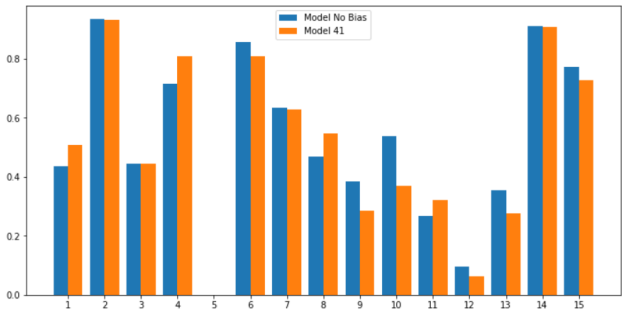


(b) Top 3 Accuracy

Figure 9: *Top 1* vs *Top 3* first class task distribution
(NB. the class are indexed starting from 0)

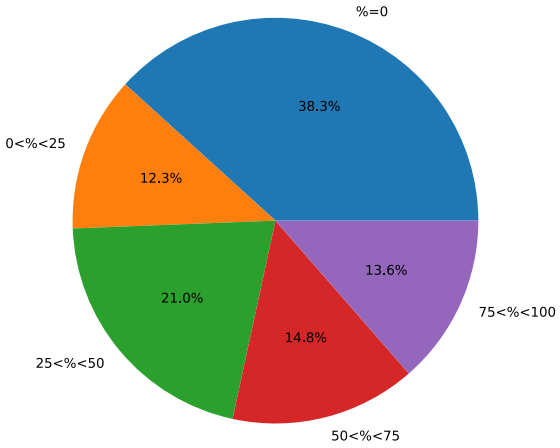


(a) First class Top 1 accuracy

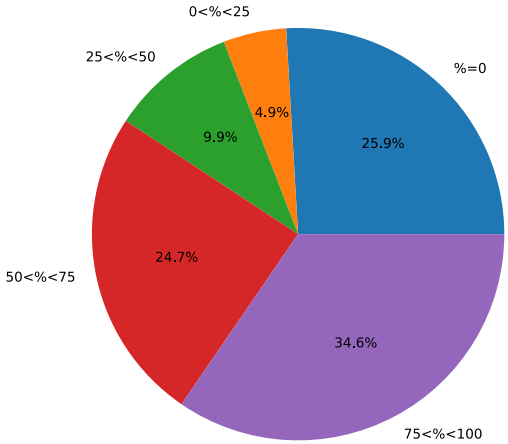


(b) Second class Top 1 accuracy

Figure 10: *First class* vs *Second class* accuracy comparison between with/out bias (see **Section 5.2.1**)



(a) Top 1 accuracy pieplot



(b) Top 5 accuracy pieplot

Figure 11: *Top 1* and *Top 5* accuracy pieplots for Second Level Task testing