Named Entity Recognition for the Romanian Language

Teofana Enăchioiu ETIC2272@SCS.UBBCLUJ.RO
Faculty of Mathematics and Computer Science
Babeş-Bolyai University
Cluj-Napoca, Romania

1. Problem statement

This project aims to apply the task of **Named Entity Recognition (*NER*)** to the Romanian language using a Deep Learning approach. More exactly, we will use a BERT model and we will fine-tune it to obtain better results.

The story of the Romanian NER is a quite recent one. This task came to the attention of the Romanian researchers in the last 2-3 years when they started to develop a data set (*RONEC*) and to develop the *Bert Base Model for Romanian*.

2. Proposed solution

The proposed solution aims to enhance the BERT model, by applying fine-tuning.

2.1 Theoretical aspects

The core of this project is the *BERT* model. It stands from *Bidirectional Encoder Representations from Transformers* and it is a recent technical innovation in applying the bidirectionality in transformers, in language modeling. It has been pre-trained on *Wikipedia* and *BooksCorpus* and it requires task-specific fine-tuning in order to be adjusted to a certain domain.

BERT uses ideas from Transformer which is an attention mechanism used for learning contextual relations between words. The transformer consists of an encoder that reads the text input and a decoder that produces a final result. From this mechanism, BERT uses only the encoder part, because its goal is to generate a language model.

The pre-training part of BERT is the key part. Two tasks are performed here. The first one is **Masking** and it consists of randomly choosing 15% of tokens and replacing them with the *[MASK]* token (80% of the time), a random token (10% of the time), or the same token (the token is unchanged in 10% of the time). The model then tries to predict the hidden token, based on the context. To perform this task we need to feed the model with the embeddings, let the encoder perform the computations, and add a classification layer on the top of the encoder to predict the results.

Because sometimes we need more context, BERT performs a second task called **Next Sentence Prediction**. This task consists of a binary classification which told us if a pair of sentences are connected. The beginning of the sentence is marked with *[CLS]* and the

text separators are marked with *[SEP]*. During the training process 50% of input are valid pairs and 50% are random pairs.

To use BERT for Named Entity Recognition is quite easy. We only need the output vectors of each token. We use these vectors to feed the classification layer that predicts the label.

In this project, we use a particular BERT model called *Bert Base Multilingual Cased*. This model was pre-trained on the top 104 languages using a masked language modeling (MLM) objective. This model is case sensitive and this means that it makes a difference between *english* and *English*. Even if there is a specialized model for the Romanian language, we will not consider it in our project. We will use *Bert Base Multilingual Cased* model because it is easier to integrate it into our algorithm.

2.2 Application

The applications of Named Entity Recognition are various. It can be used in question answering, machine translation, or automatic text summarization. For this project, we built a simple application in which the user types a sentence and the algorithm displays the text with words annotated. Our entities of interest are colored red.

The application is very simple and the class diagram is not relevant, because we have only two classes (for GUI and prediction). However, what is important is the sequence diagram. Figure 1 illustrates this diagram and Figure 2 presents the graphical interface of our application.

3. Implementation details

More implementation details can be found in the second PDF file, which is the source code of the NLP task. **There we explained each step and also provided the result of every computation**. Figure 3 presents the flowchart of the purposed approach and highlights the most important steps.

The training part was performed in Google Colab using GPU and CUDA. We use PyTorch for the Named Entity Recognition task and WxPython for the application graphical interface. The source code of the GUI and prediction algorithm can be found in files *gui.py* and *ner_model.py*.

Our contribution in this project consists of the fine-tuning part of the algorithm. Also, applying the NER to the Romanian language is another important aspect, because this task is quite unexplored.

4. Experiments and results

For the experimental part, we tried to compare two types of fine-tuning. The first one implies full fine-tuning. We add some weight decay as regularization to the main weight matrices. The second approach consists of just train the linear classifier on top of BERT and keep all other weights fixed. Figure 4 pictures the source code of these two approaches.

Table 1 and Table 2 present the results of applying the fine-tuning approaches. The other configurations in the algorithm are the same. As it can be seen, the first approach

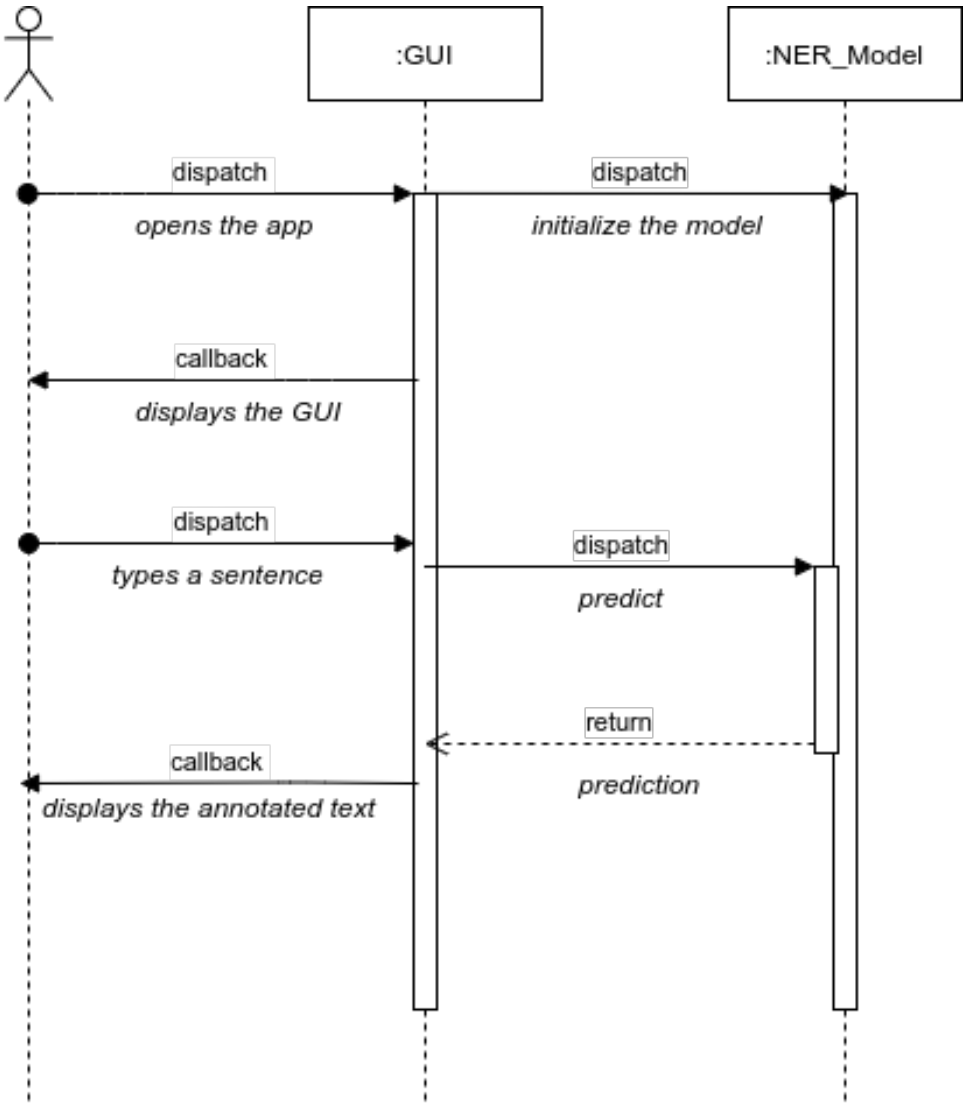

Figure 1: The sequence diagram of the application

tends to decrease. The result of the second approach does not change very much during the epochs.

Epoch	Train loss	Valid loss	Valid accuracy	Valid F1 score
1	0.746	0.377	0.896	0.733
2	0.310	0.300	0.915	0.789
3	0.230	0.295	0.915	0.794

Table 1: First fine tuning approach

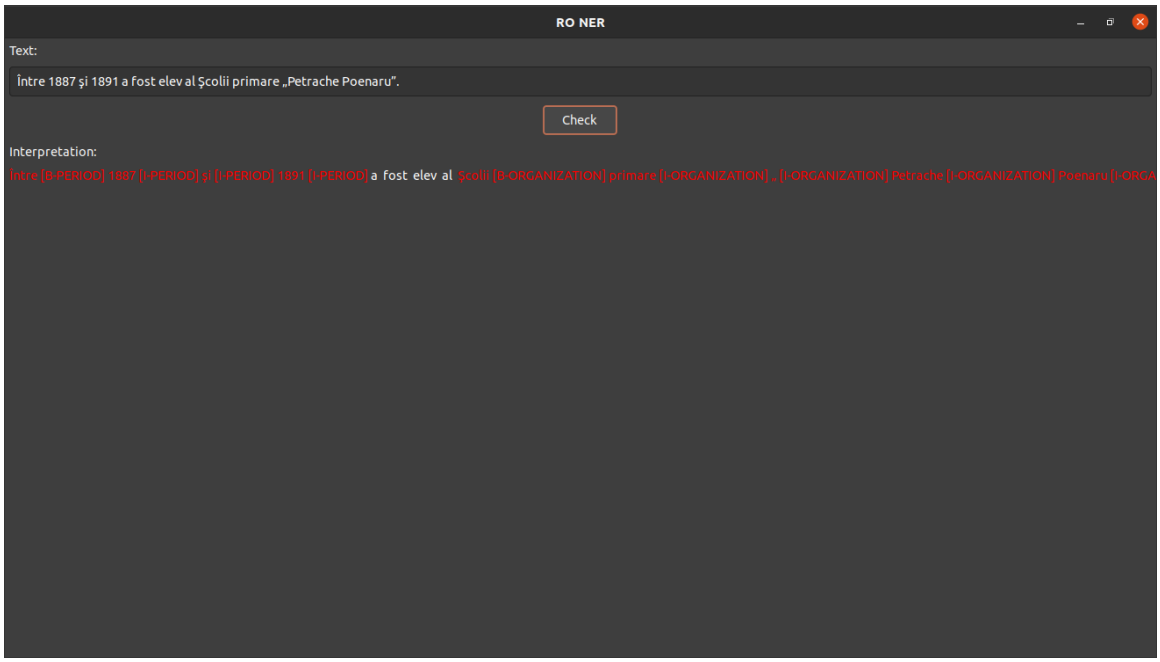

Figure 2: The graphical interface of the application

Epoch	Train loss	Valid loss	Valid accuracy	Valid F1 score
1	0.204	0.2951	0.9166	0.7957
2	0.199	0.2956	0.9163	0.7957
3	0.199	0.2958	0.9163	0.7956

Table 2: Second fine tuning approach

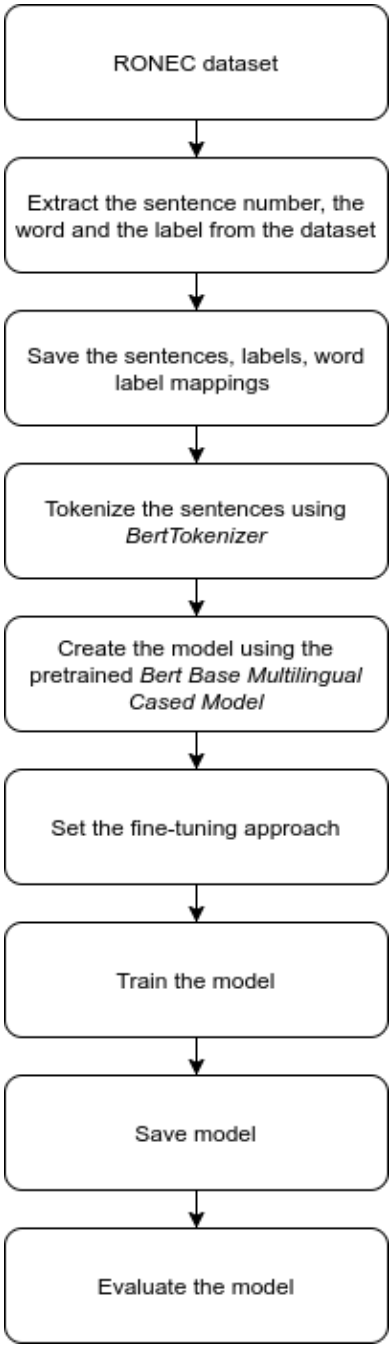

Figure 3: Flowchart of the purposed approach

```
full_fine_tuning = False
if full_fine_tuning:
    param_optimizer = list(model.named_parameters())
    no_decay = ['bias', 'gamma', 'beta']
    optimizer_grouped_parameters = [
        {'params': [p for n, p in param_optimizer
                    if not any(nd in n for nd in no_decay)],
         'weight_decay_rate': 0.01},
        {'params': [p for n, p in param_optimizer
                    if any(nd in n for nd in no_decay)],
         'weight_decay_rate': 0.0}
    ]
else:
    param_optimizer = list(model.classifier.named_parameters())
    optimizer_grouped_parameters = [{"params": [p for n, p in param_optimizer]}]
```

Figure 4: Fine tuning approaches

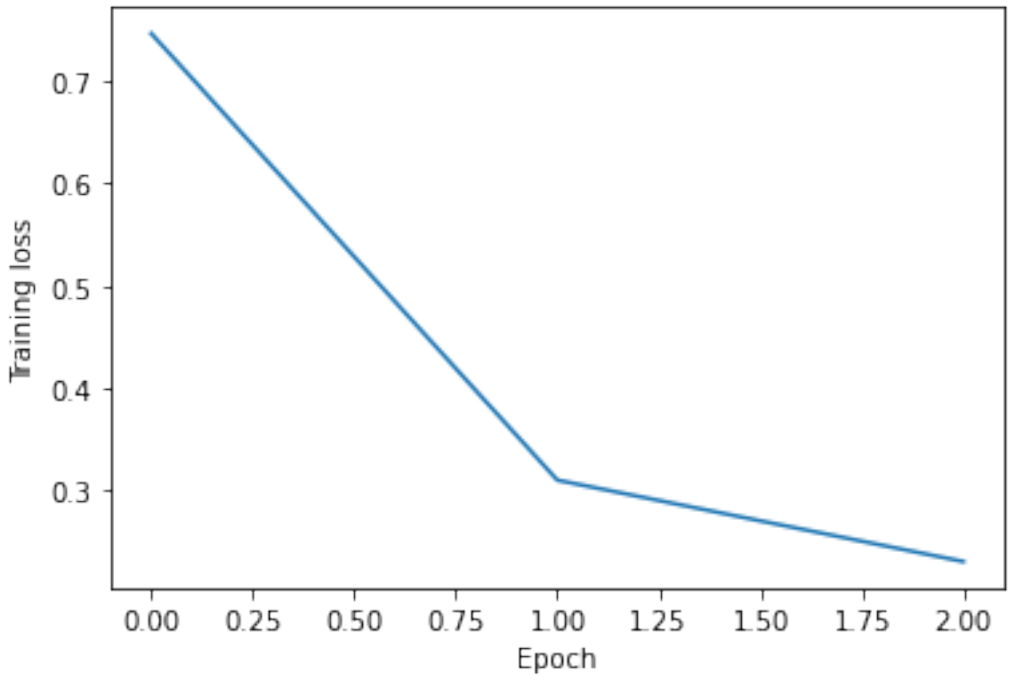

Figure 5: First fine tuning approach - Training loss

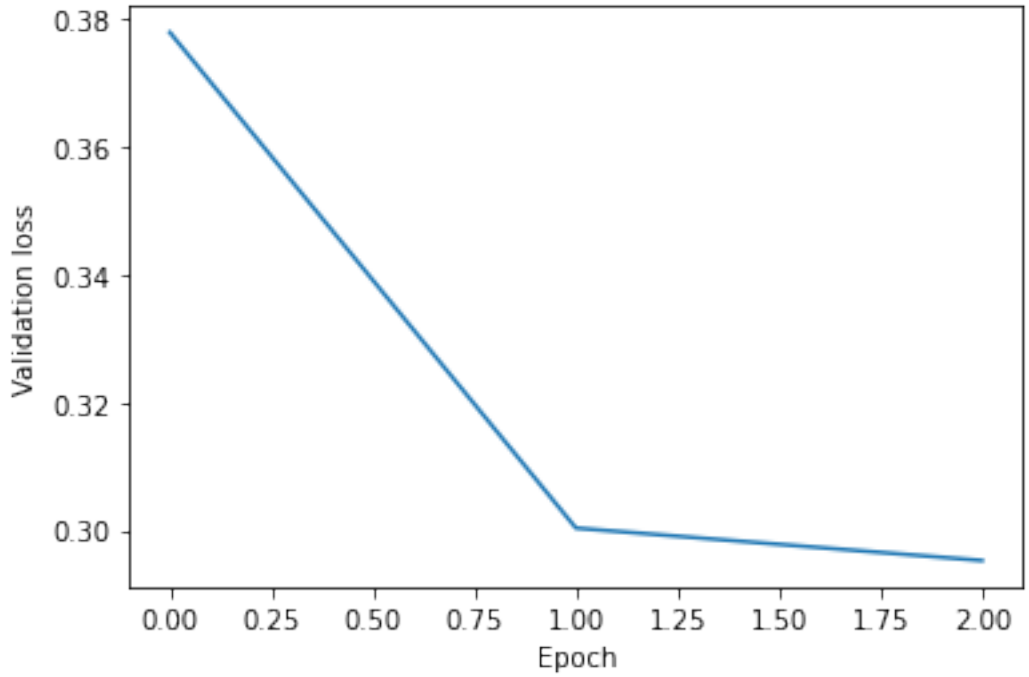

Figure 6: First fine tuning approach - Validation loss

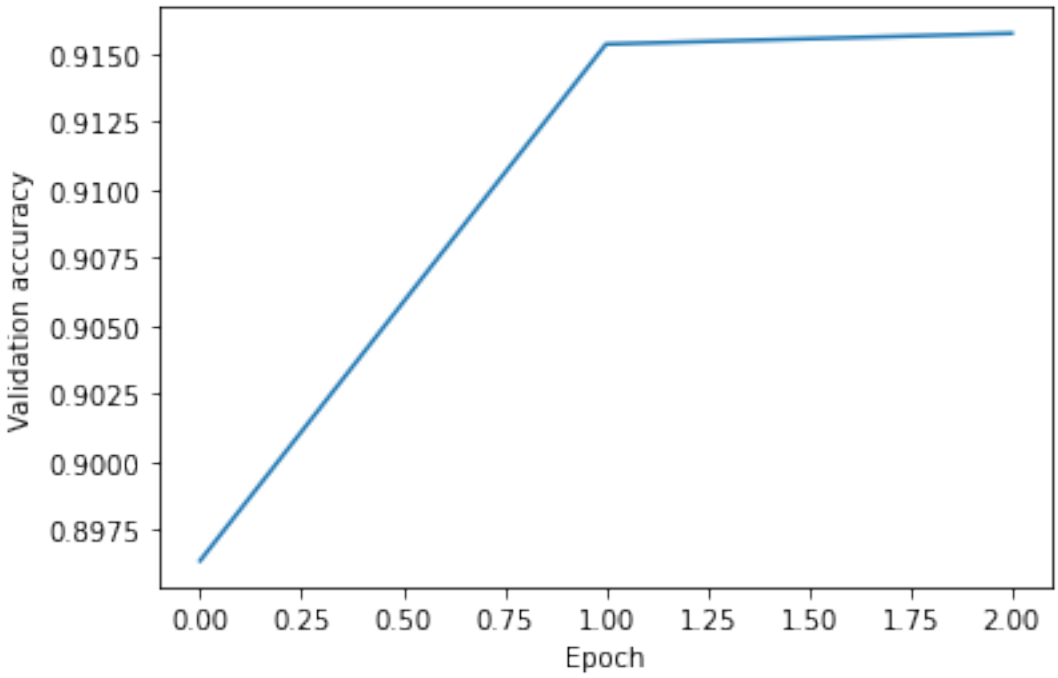

Figure 7: First fine tuning approach - Validation accuracy

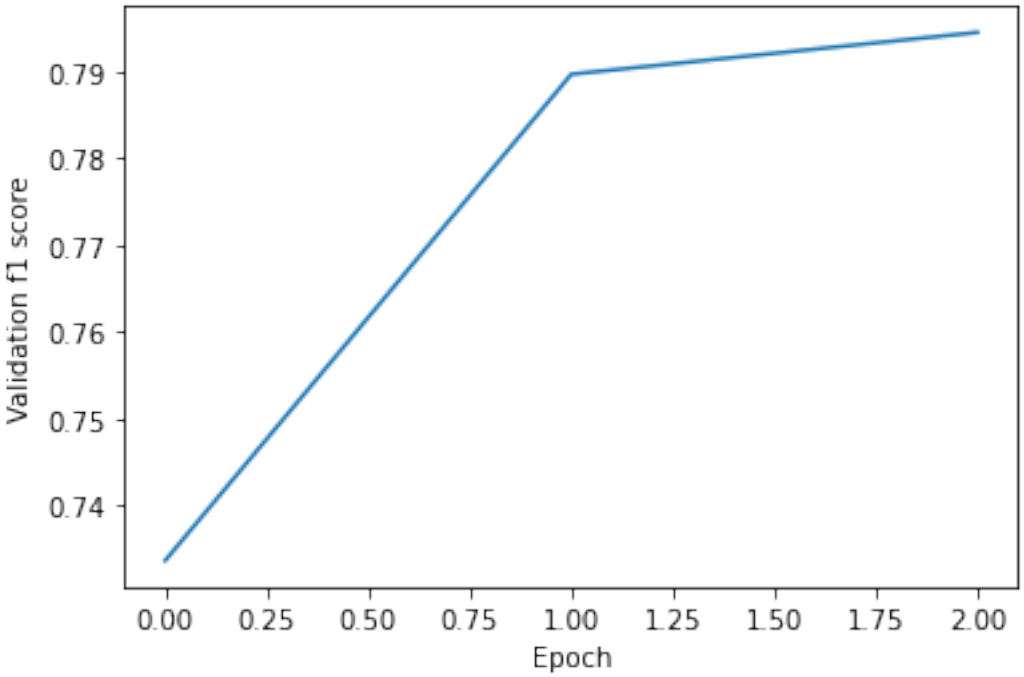

Figure 8: First fine tuning approach - Validation F1 score

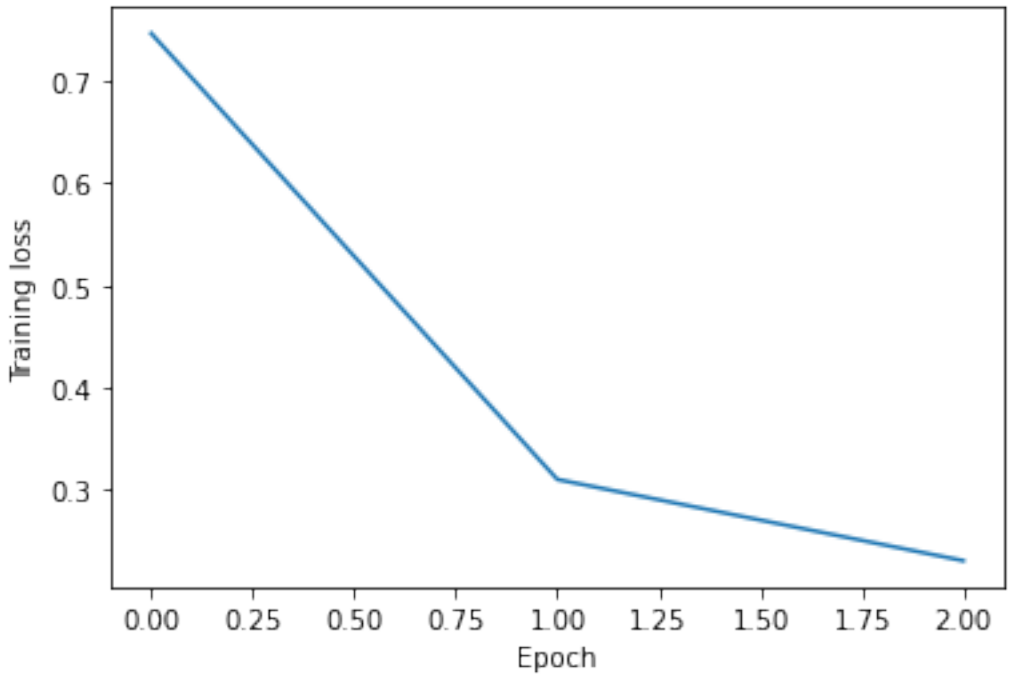

Figure 9: Second fine tuning approach - Training loss

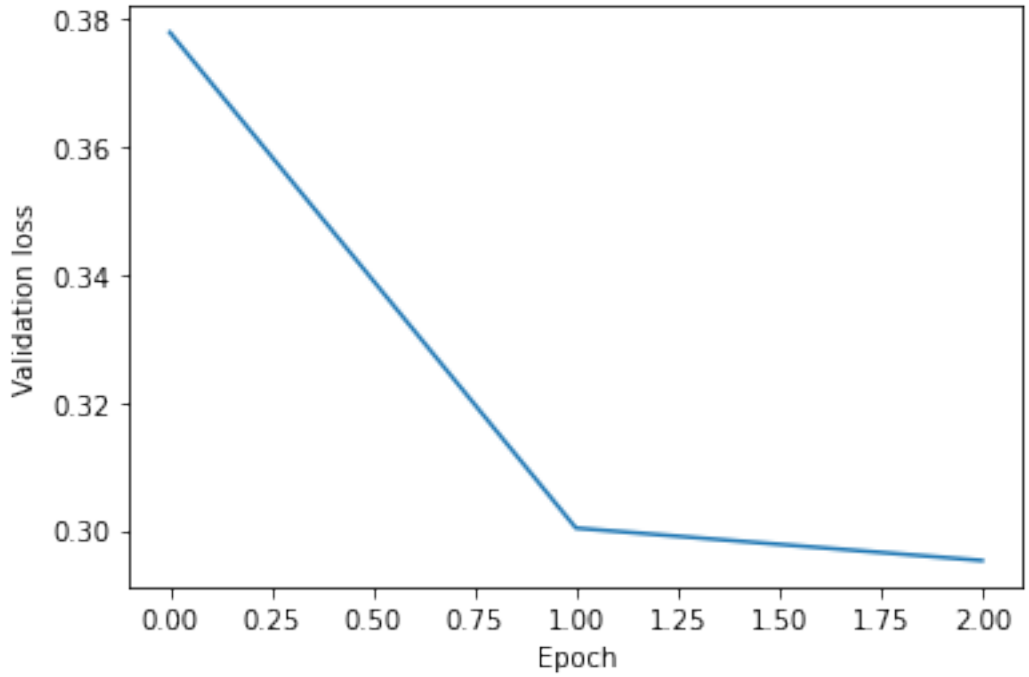

Figure 10: Second fine tuning approach - Validation loss

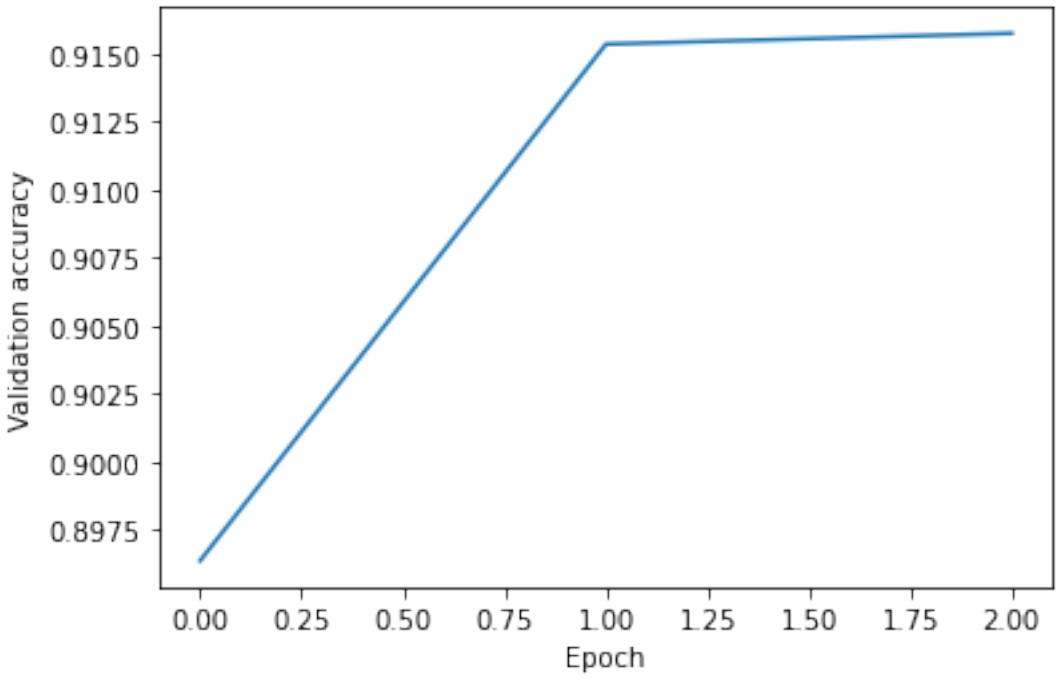

Figure 11: Second fine tuning approach - Validation accuracy

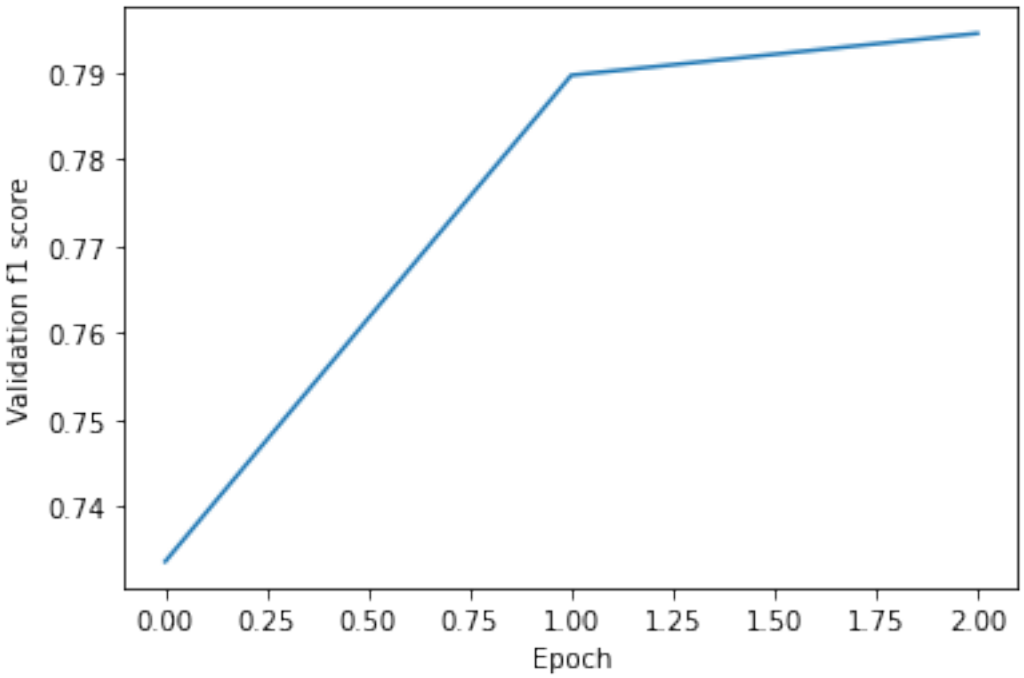

Figure 12: Second fine tuning approach - Validation F1 score