Finetuning pretrained models for Natural Language Inference

Hung Huu Hoang
Department of Computer Science
University of Western Ontario
London, Ontario
hhoang26@uwo.ca

***Abstract*—For humans, it is not so difficult to tell if a statement A implies statement B. For machines, however, this remains a challenging task. In this project, I explore effects of finetuning pretrained language models for the task of multilingual Natural Language Inference via a Kaggle Competition. I experimented with finetuning pretrained multilingual versions of BERT, Deberta-V3 and Flan-T5. The results showed that finetuned Deberta-V3 Base model produced the best Test result, even better than a finetuned larger Flan-T5 model, and with a rather consistent performance across languages. The best accuracy of 81.8% was obtained via an ensemble model.**

***Index Terms*—NLI, multilingual, pretrained models, finetuning, ensemble, mBERT, mDeberta-V3, Flan-T5, Kaggle**

I. Introduction

In this Kaggle Competition named ”Contradictory, My Dear Watson” [1], the Training Set consists of pairs of sentences, sentence A and sentence B, and the task is to decide if the content in sentence B is implied, contradicted or ”neither” by sentence A. Here, “neither” means the content in B is neither suggested nor refuted by the content in A.

In Natural Language Processing (NLP) literature, this kind of tasks is referred to as a Natural Language Inference (NLI) task because it involves some kind of deduction or inference. NLI tasks help assess whether the system really “understands” what is mentioned in sentence A or not by asking it to tell if another piece of information (sentence B) can be deduced from it. One aspect of this competition that makes it quite challenging is that it contains data written in multiple languages, not just in English.

Pretrained language models have become the defactor starting points for various NLP tasks, including NLI, as they have been shown to be able to capture the linguistic aspects of language. In this project, I experimented with finetuning some well-known pretrained models whose multilingual pretrained models are available, including: BERT, DebertaV3 and Flan-T5 [1]. I used the pretrained models made available on HuggingFace [2].

The results showed that among single finetuned models, finetuned mDeberta-V3 gave the best accuracy of 75.5% on the Test Set, surpassing all other models, including finetuned Flan-T5 Large.

Analyzing performance of the finetuned models over specific languages showed some surprising results and helped identifying an ensemble method that lead to the best Test-Set performance of 81.8%.

[1]In addition to the above 3 model families, I also experimented with the multilingual version of Roberta. But for some reasons still unknown, the finetuning of this model did not work. As such, this model is omitted from the list above.

II. Background on pretrained models finetuning

The introduction of the Transformer architecture in 2017 [3] and the successful application of this model in pretraining BERT [4] have opened a new wave of transfer learning via pretrained language models. Pretrained language models are language models trained on enormous datasets containing billions of words, and often trained on various language-related tasks, most often the masked tokens prediction task, as used in BERT. As these models are able to capture the linguistic aspect of languages, from syntax to semantics, they serve well as base models for diverse downstream tasks.

Given a pretrained model, finetuning it for downstream tasks means retraining it on a specific dataset for a target task. Before this retraining is carried out, the pretrained model is usually put on top (at the last-most Transformer block) one or a few fully connected layers, although this can be any other types of layers to suit the goal, where the last of these new layers is designed to have the desired number of output neurons suiting the target task. For NLI, this last layer will contain 3 output logits for each of the 3 classes (Entailment, Contradiction and Neutral). In terms of the data required for finetuning, the target task's dataset can be quite small, starting from just tens, but more commonly hundreds, of training examples. Compared to pretraining, finetuning is much easier and faster to carry out as it requires substantially less training time and compute power (usually a single GPU or TPU would suffice).

In standard finetuning, all the parameters of the pretrained models and the additional layers put on top are updated during the training process on the target dataset. This is also the approach I used in this project. Due to limited computational resources, the largest pretrained model that I could finetune was Flan-T5 large with 780 million parameters.

III. Kaggle Dataset descriptions

For this competition, we are given 2 datasets: Train and Test datasets. The Train and Test Set contains 12120 and 5195 examples, respectively.

Each example consists of a pair of sentences and other fields in a comma-separated format as follows:

- id: a unique string representing this example.
- premise: sentence A.
  Example: *"and these comments were considered in formulating the interim rules."*
  Notes: As can be seen from this example, although we call it a "sentence", it may not be a full linguistic sentence, and may be just a fragment of a sentence. It nevertheless contains sufficient information for our inference task.
- hypothesis: sentence B.
  Example: *"The rules developed in the interim were put together with these comments in mind."*
- lang_abv: the abbreviation code of the language.
  Example: *"en"*
- language: the language of this pair of sentences.
  Example: *"English"*
- label: either *0*, *1* or *2*, indicating the relationship between sentence A and sentence B, as follows:
  - 0: indicates entailment: B is implied by A
  - 2: indicates contradiction: B is contradicted by A
  - 1: indicates neutrality: when neither of the above 2 applies.

  In the example pair given above, the label would be *0* to indicate entailment.

The Train and Test datasets in this Kaggle Competition contain pairs of sentences written in 15 languages: Arabic, Bulgarian, Chinese, English, French, German, Greek, Hindi, Russian, Spanish, Swahili, Thai, Turkish, Urdu and Vietnamese. Fig. 1 shows the composition of these languages in the training data set.

Among these 15 languages, English is the predominant language, comprising 56.7% of the entire training data set. The remaining 14 languages are quite equally distributed, each contributes about 3% of the data set.

In terms of distribution of classes, the Training Set contains a balance of the 3 classes. Fig. 2 shows the number of examples and percentage of each class.

For the Test Set, as they are meant only for actual predictions and rankings in the competition's Leaderboard, I did not look at nor analyze it. But the assumption here is that the majority of the Test data would follow a similar distribution as that of the Training data.

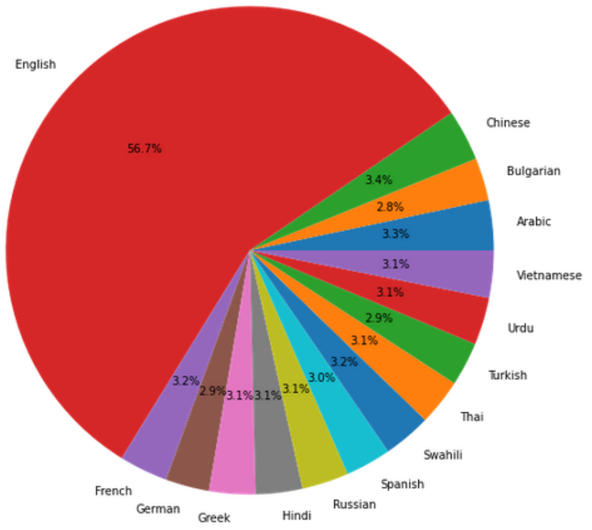

Fig. 1. Distribution of languages in the Training Set

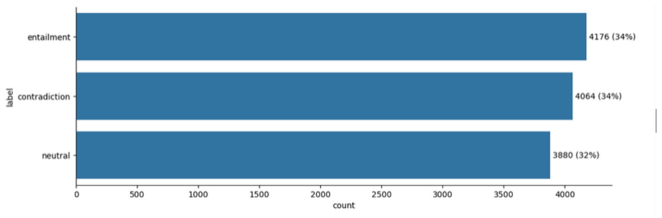

Fig. 2. Distribution of target classes in the Training Set

IV. Methodology

A. Finetuned models

In this project, I experimented with finetuning the following 3 multilingual pretrained model families for this Kaggle competition. These models were chosen because they are small enough for finetuning on Google Colab [5] and their pretrained models are publicly available. The pretrained models I used were from HuggingFace:

- BERT multilingual Base cased (mBERT) [4], [6]
- Deberta V3 multilingual Base (mDeberta-V3) [7], [8]
- Flan-T5 Base and Large versions. [9]–[11]

All of the 15 languages in this Kaggle competition are supported by these multilingual models.

In terms of architecture, these models all have Transformer as their underlying building blocks, but they differ in whether they are Encoders, Decoders or both. Their pretraining process and task are also different, but a more detailed description of their differences are beyond the scope of this project. Table I summarizes key aspects that are of relevance in this project.

TABLE I
KEY ASPECTS OF FINETUNED MODELS

Model	Languages	Architecture	Parameters
mBERT Base	104	Encoder	178M
mDeberta V3 Base	102	Encoder	279M
Flan-T5 Base	60	Encoder-Decoder	250M
Flan-T5 Large	60	Encoder-Decoder	780M

B. Evaluation Metric

The evaluation metric used in this Kaggle Competition is accuracy: the number of instances with correct predicted class over all instances. For this Kaggle Competition and this project, we are not concerned with accuracy per class but the overall accuracy across all 3 classes, as the classes are equally represented in the data.

Accuracy over the Dev Set can be calculated based on the provided class labels. For the Test Set, whose class labels are not provided to participants, we need to upload our predicted class for each instance to Kaggle and receive an accuracy number returned.

C. *Finetuning details*

For finetuning, we need to have a Dev Set to detect overfitting and stop at the first 2 checkpoints whose Dev loss continued to increase, i.e., using early stopping. For this purpose, I took 20% of the data from the provided Train Set to form my Dev Set. I first randomized the training data and taking 20% stratified by labels (the target class) to make sure the Dev result accurately reflects the distribution of the 3 NLI classes in our training data.

There are a couple of important hyper-parameters that may affect finetuning process. Table II summarizes values of the parameters used in the finetuning process. They were used for all finetuned models.

TABLE II
VALUES OF KEY HYPER-PARAMETERS FOR FINETUNING

Hyper-parameter	**Value**
Learning rate	5e-05
Warmup ratio	0.1% of training steps
Optimizer	AdamW
Weight decay	0.01
Batch size	16 (6 for Flan-T5 Large)
Gradient accummulation steps	1 (3 for Flan-T5 Large)
Loss function	Cross-entropy loss

The optimizer used (also by default in the HuggingFace Library) is AdamW [12], which applies the weight decay directly to the weight update steps, instead of at the loss calculation step. The warmup ratio, a percentage of total training steps, indicates the number steps over which the learning rate is linearly increased from 0 up to the specified learning rate. Gradient accummulation steps refers to the number of forward steps after which accumulated gradients are averaged and a backward pass is carried out to update the weights.

All of these models were finetuned on Google Colab using a V100 GPU in high-ram mode, except for Flan-T5 Large where V100 GPU would run out of memory. For this model, I needed to use the TPU option, slower but with higher memory, for finetuning.

For all models except Flan-T5 Large, it took < 1 hours to complete the finetuning (i.e., being able to identify the best epoch). For Flan-T5 Large, it took about 4 hours per epoch. These were the time needed for a successful continuous run, assuming no other issues that may happen on Google Colab such as having the runtime disconnected or issues related to models storage (required storage for each saved checkpoint ranged from 2GB for BERT up to 9GB for Flan-T5 Large).

V. EXPERIMENTS AND RESULTS

A. *Results of finetuned models*

The following tables (Table III to Table VI) show the loss on the Train and Dev Sets as well as Accuracy on Dev Set for each finetuned model. For all models except Flan-T5 Large, results were recorded after each epoch. For Flan-T5 Large, results were recorded after every 300 backward steps. The best model as evaluated on the Test set corresponds to the row containing the Dev loss and Accuracy in bold.

TABLE III
MBERT BASE FINETUNING RESULTS

Epoch	**Train loss**	**Dev loss**	**Dev Accuracy**
1	0.980	0.866	61.6
2	0.780	**0.783**	**66.3**
3	0.559	0.980	66.9
4	0.264	1.245	66.8

TABLE IV
MDEBERTA-V3 BASE FINETUNING RESULTS

Epoch	**Train loss**	**Dev loss**	**Dev Accuracy**
1	0.882	0.614	76.6
2	0.555	**0.632**	**77.3**
3	0.374	0.787	78.3
4	0.237	1.126	78.2

TABLE V
FLAN-T5 BASE FINETUNING RESULTS

Epoch	**Train loss**	**Dev loss**	**Dev Accuracy**
1	0.215	0.169	68.3
2	0.180	0.176	68.6
3	0.156	0.172	70.0
4	0.152	**0.184**	**70.0**
5	0.133	0.187	69.6

TABLE VI
FLAN-T5 LARGE FINETUNING RESULTS

Step	**Train loss**	**Dev loss**	**Dev Accuracy**
300	N/A	0.156	71.9
600	0.160	**0.152**	**73.5**
900	0.160	0.158	73.2
1200	0.136	0.167	73.5
1500	0.105	0.184	72.7

The Test results of these finetuned models are shown in Table VII.

B. *Compare zero-shot and finetuned performance*

To see if and by how much these finetuned version perform better than pretrained models, I also evaluated zeroshot performance of these models over the Test Set. Table VIII compare zero-shot and finetuned performance of these models.

From table VIII, we can see the power of finetuning: finetuning helped increase about 10% of Test accuracy for Flan-T5

TABLE VII
DEV & TEST RESULTS OF FINETUNED MODELS

Finetuned Model	Dev Accuracy	Test Accuracy
FT mBERT Base	66.3	65
FT mDeberta-V3 Base	77.3	75.5
FT Flan-T5 Base	70.0	69.2
FT Flan-T5 Large	73.5	73.3

TABLE VIII
COMPARE TEST-SET ZERO-SHOT AND FINETUNED PERFORMANCE

Model	Zero-shot	Finetuned
mBERT Base	32.3	65
mDeberta-V3 Base	33.6	75.5
Flan-T5 Base	58.6	69.2
Flan-T5 Large	68.9	73.3

Base and 4.5% for Flan-T5 Large. For mBERT and mDeberta-V3, finetuning increased 32.7% and 41.9% respectively.

For mBERT and mDeberta-V3, they were not pretrained on NLI-related tasks, and this explained why zero-shot performance was very low, just about the same as a random guess. After finetuned, however, their performance increased dramatically, with mDeberta-V3's finetuned performance even exceeded that of Flan-T5 Large's.

For Flan-T5's, as they were pretrained on more than 1000 tasks, including NLI tasks, their zero-shot performances were well above 50%. Yet, we can still see clear improvements when finetuned over this Kaggle Dataset.

Another observation from the above results is that the effect of finetuning became less significant for the Large version compared to the Small version of Flan-T5. Although I did not experiment with larger Flan-T5 models, i.e., Flan-T5 XL and Flan-T5 XXL, due to resource constraints, my educated estimate is that, in the same trend with the results of the Base and Large Flan-T5 models shown above, the impact of finetuning would diminish as pretrained models grow bigger.

C. *Ensemble of the best finetuned models*

In order to see if I could increase the accuracy by combining these finetuned models, I experimented with creating an ensemble from the 3 best performing finetuned models: mDeberta-V3, Flan-T5 Large and Flan-T5 Base.

This combination method was used: if mDeberta's classification result was different from that of Flan-T5 Large, I would use Flan-T5 Base's output as the deciding vote. In the case that Flan-T5 Base's predicted class was not the same as any of mDeberta's and Flan-T5's, mDeberta's predicted class was chosen, as mDeberta was the best performing model, based on Table VII.

Results of this ensemble is shown in Table IX.

TABLE IX
ENSEMBLE OF MDEBERTA AND FLAN-T5 MODELS

Model	Dev Accuracy	Test Accuracy
Ensemble	76	76.2

This ensemble performed only slightly better than mDeberta-V3, which scored a 75.5% accuracy over the Test Set. This result seemed to indicate that something was amiss, and so I decided to look at the performance of these 3 models over each of the 15 languages in our Dev Set.

D. *Performance break-down for each language*

As the Dev Set provides language annotation (field "language") for each training example, performance for each language could be calculated for each model. Table X, XI, XII, XIII shows performance of finetuned mBert, finetuned mDeberta-V3, Flan-T5 Base (finetuned and zero-shot) and Flan-T5 Large (finetuned and zero-shot) respectively. For all tables, results are sorted by Dev Accuracy.

TABLE X
FINETUNED MBERT DEV-SET RESULT FOR EACH LANGUAGE

Language	Dev Accuracy
Spanish	71.3
English	69.7
Bulgarian	67.9
German	67.2
Arabic	66.3
French	66.2
Vietnamese	64.7
Urdu	61.8
Chinese	61.5
Russian	59.5
Turkish	59.5
Greek	59.2
Hindi	57.9
Thai	55.8
Swahili	51.9

TABLE XI
FINETUNED MDEBERTA-V3 DEV-SET RESULT FOR EACH LANGUAGE

Language	Dev Accuracy
Bulgarian	85.7
Spanish	82.5
Arabic	82.0
Turkish	79.7
English	79.3
Greek	76.3
German	76.1
Hindi	75.4
Russian	74.3
French	72.3
Vietnamese	72.1
Chinese	70.5
Thai	69.8
Urdu	67.4
Swahili	64.6

Many interesting and surprising information could be observed from these break-down tables about each model. Table X and Table XI shows that for both finetuned mBERT and finetuned mDeberta-V3, their performance across languages were rather consistent, with lowest accuracy for mBERT was 51.9 (for Swahili) and for mDeberta-V3 was 64.6 (for Swahili). In stark contrast, Flan-T5 models, even after finetuned, produced very different performances for different languages, with the

TABLE XII
FLAN-T5 BASE DEV-SET RESULT FOR EACH LANGUAGE

Language	Finetuned	Zero-shot
English	85.5	73.6
Spanish	75	62.5
German	74.6	56.7
French	64.6	52.3
Bulgarian	64.3	57.1
Turkish	59.5	47.3
Russian	55.4	44.6
Vietnamese	45.6	23.5
Arabic	44.9	31.5
Swahili	44.3	40.5
Greek	40.8	30.3
Hindi	40.6	37.7
Chinese	34.6	33.3
Thai	33.7	31.4
Urdu	30.3	43.8

TABLE XIII
FLAN-T5 LARGE DEV-SET RESULT FOR EACH LANGUAGE

Language	Finetuned	Zero-shot
English	88.9	85.1
German	83.6	74.6
Spanish	81.3	70
Bulgarian	80.4	55.4
French	80.0	63.1
Turkish	68.9	62.2
Russian	64.9	55.4
Swahili	46.8	40.5
Arabic	41.6	30.3
Vietnamese	41.2	33.8
Hindi	37.7	40.6
Greek	36.8	35.5
Thai	36.0	37.2
Chinese	30.8	30.8
Urdu	25.8	42.7

most notable point being that for a few languages, their accuracies are just about a random model, i.e., in the range [30%, 40%].

Looking at the effect of finetuning (vs zero-shot) for Flan-T5 models from Table XII and Table XIII, it could be observed that, apart from some minor degradation, performance increased across languages after finetuning. This once again confirmed the benefit of finetuning, even for all 14 non-English languages where their training data accounts for only about 3% of the total Train Set, which means on average just about 300 examples for finetuning.

Another very important point that can be observed from Table XIII is that for finetuned Flan-T5 Large, there were only 3 languages where it outperformed finetuned mDeberta-V3: English (88.9 vs 79.3), German (83.6 vs 76.1) and French (80.0 vs 72.3). For all remaining 12 languages, mDeberta-V3 outperformed Flan-T5 Large, and for many of them, by a large margin, with the most extreme case being that of Urdu language where mDeberta-V3 obtained an accuracy of 67.4 vs 25.8 of finetuned Flan-T5 Large. This observation suggested a promising way to ensemble these models to boost performance, which is discussed in the next subsection.

E. *Ensemble by language-based model selection*

Based on our discussion above about different performances of the finetuned models for different languages, the following ensemble method becomes evident: for all languages, use finetuned mDeberta-V3, except for English, German and French where finetuned Flan-T5 Large was a better choice. In this combination, Flan-T5 Base was no longer used.

I note here again that for this specific Kaggle dataset, language annotations were available for both the Train and Test Sets. This information facilitated our models selection step above, which otherwise would have required access to a small model for language detection.

Using this ensemble method, Table XIV shows our best result in this project.

TABLE XIV
LANGUAGE-BASED ENSEMBLE OF MDEBERTA-V3 AND FLAN-T5 LARGE

Model	Dev Accuracy	Test Accuracy
Ensemble	83.1	81.8

VI. CONCLUSIONS AND FUTURE WORK

In this project, I worked on the multilingual NLI task introduced in the Kaggle Competition called ”Contradictory, My Dear Watson”. The approach I took to tackle this problem was finetuning the following pretrained multilingual models: mBERT Base, mDeberta-V3 Base and Flan-T5 (Base and Large versions).

Performance of finetuned models, as discussed above, demonstrated the effectiveness of finetuning, across all 15 languages, even when the number of finetuning examples for all 14 non-English languages was just about 300 on average.

Among the above models, my experiments indicated the superior performance of finetuned mDeberta-V3 Base over this Kaggle multilingual dataset, especially in comparison with finetuned Flan-T5 Large, even though Flan-T5 Large has almost 3 times the number of parameters. Another noteworthy observation was that finetuned mDeberta-V3's performance across languages was very consistent, which was something finetuned Flan-T5 Large could not achieve.

My best result was obtained via an ensemble between finetuned versions of mDeberta-V3 Base and Flan-T5 Large where Flan-T5 Large was used for Engish, German and French and mDeberta-V3 was used for all remaining languages. The best Test accuracy obtained was 81.8%.

In this project, I only used the dataset provided by this Kaggle Competition for finetuning, which was quite small (12K examples). This made my results on these 15 languages non-comparable to other published results, especially those from mT5 [13], a T5-based multilingual model. In their experiments, they finetuned mT5 on XNLI [14], a multilingual dataset that is much larger with about 400K training examples per language. Their results indicated improved performance across languages after finetuning. For future work, it would be interesting to repeat the experiments that have been done in this project for mDeberta-V3 and Flan-T5, but using the larger XNLI dataset.

References

[1] P. C. Amy Jang Ana Sofia Uzsoy. "Contradictory, my dear watson — kaggle." (2020), Available: https://kaggle.com/competitions/contradictory-my-dear-watson.

[2] HuggingFace. "Models - huggingface," Available: https://huggingface.co/models.

[3] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] Google. "Google colab," Available: https://colab.research.google.com.

[6] HuggingFace. "Bert-base-multilingual-cased - huggingface," Available: https://huggingface.co/bert-base-multilingual-cased.

[7] P. He, J. Gao, and W. Chen, "Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing," *arXiv preprint arXiv:2111.09543*, 2021.

[8] HuggingFace. "Microsoft/mdeberta-v3-base - huggingface," Available: https://huggingface.co/microsoft/mdeberta-v3-base.

[9] H. W. Chung, L. Hou, S. Longpre, *et al.*, "Scaling instruction-finetuned language models," *arXiv preprint arXiv:2210.11416*, 2022.

[10] HuggingFace. "Google/flan-t5-base - huggingface," Available: https://huggingface.co/google/flan-t5-base.

[11] HuggingFace. "Google/flan-t5-large - huggingface," Available: https://huggingface.co/google/flan-t5-large.

[12] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[13] L. Xue, N. Constant, A. Roberts, *et al.*, "Mt5: A massively multilingual pre-trained text-to-text transformer," *arXiv preprint arXiv:2010.11934*, 2020.

[14] A. Conneau, G. Lample, R. Rinott, *et al.*, "Xnli: Evaluating cross-lingual sentence representations," *arXiv preprint arXiv:1809.05053*, 2018.