CIL 2022: Sentiment Analysis on Tweets

Team AntipathyGlance

Klim Troyan Mengtao Zhang Fabian Landwehr Rolando Grave de Peralta

{ktroyan, mezhang, fabianla, rgrave}@student.ethz.ch

ABSTRACT

In this report, we present our approaches to obtain a high score on the two-label sentiment classification problem for tweets. We explain our understanding on this problem, our analysis on the underlying dataset, as well as the data pre-processing performed, evaluation modes used and models created. We then present our implemented baselines, fine-tuned pre-trained Transformer models, and two modified more models based on modern deep learning techniques. Finally, we present the performance of the models and the best results obtained on our own validation set as well as on the Kaggle test set. We achieve an accuracy of 0.91760 on the test set (public leaderboard).

1 INTRODUCTION

Sentiment analysis is one of the major tasks in NLP [1]. In its core, the goal of sentiment analysis is to classify for a given text whether it has a positive, neutral or negative sentiment. In our case, we only consider positive and negative sentiment. Sentiment analysis can be used to understand the public opinion different topics, understand market trends, and help governments and companies guide their strategy [2]. Furthermore, it plays an important role in human-machine interfaces [3] and medical applications [4]. Especially the short-form message platform Twitter is an interesting source of data for sentiment analysis. The reason is its popularity and huge number of available tweets that cover most topics of interest. The objective of this project is to perform sentiment analysis on a dataset of 2.5 million tweets.

2 RELATED WORK

Sentiment analysis is a thoroughly studied problem with a wide range of proposed solutions, for example [5] [6] and [7]. In particular sentiment analysis on tweets has had a comeback in the last few years due to the COVID-pandemic. For example, in the paper [8], Kamaran H. Manguri et. al, leveraged tweets about COVID-19 to better understand the spread of the disease and the sentiment toward it. This is only one of many examples. In earlier research, researchers leveraged linguistic knowledge to create features from text and then trained a binary classifier for the predictions [9]. However, in more recent research the best-performing models no longer use such custom features. Instead, huge, pre-trained transformer-based models that are fine-tuned to the task are the state of the art [10] [11]. In particular, many BERT-based [12] models such as roBERTa [13], deBERTa [14], XML-roBERTa, [15] can be fine-tuned to achieve a high accuracy on Twitter sentiment analysis tasks.

3 METHODS

3.1 Data

3.1.1 Exploratory Data Analysis. Before creating baselines and high-performing models, we explored the given data. We immediately observe that the predictions should either be positive or negative, without a possible neutral label that is usually the dominant sentiment. However, when looking at the data as a human, it is obvious that many of the tweets are actually neutral (neither positive nor negative). On further inspection, many of the tweets are "wrongly" labeled. To understand why this is the case, we should know how the dataset was created. The dataset is only composed of tweets that contained one of the following strings of characters: ":)", or ":(". During the creation of the datasets that we were given, the aforementioned strings were removed and the samples were attributed the label "0" for a negative smiley and "1" for a positive one. As an example of a "mislabelled" tweet in the dataset, we can consider the following: "expo precision point whiteboard eraser (8 473kf) (office product ergonomic eraser with designated grip . precisi ... <url>". It is classified as negative although it would clearly be considered neutral by a human and thus would likely be the intent of the entity that tweeted this. The reason becomes clear when looking at the corresponding tweet on Twitter, which is: "Expo Precision Point Whiteboard Eraser (8473KF) (Office Product): Ergonomic eraser with designated grip. Precisi... [amazon link]". It can be seen that it contains a the string "):", and was hence classified as negative. If the goal is to represent a truthful sentiment classification, this is an important caveat of distant supervision method applied for the creation of this dataset. In fact, a substantial part of the dataset consists of such tweets. Thus, even large Transformer models, pre-trained on hundreds of millions of tweets, need to be fine-tuned to the peculiarities of this dataset to perform well. While there would still be concepts such as sarcasm to consider, this most likely reduced the quality of the results that could have been obtained if only the tweets containing "real" (i.e., the user's intent was to use a string ":)" or ":(" to represent a smiley) smileys.

In addition, we noticed that the test set contains a substantial number of duplicated tweets. There is a tweet that appears as many as 168 times [16]. From this, we infer that these tweets are of high importance regarding the test evaluation score. Hence, in a real-life application, it would make much sense to ensure that they are labeled correctly. For example, one could define a threshold $t = 6$ and ensure that all tweets with $\geq t$ occurrences in the test data are labeled correctly through a secondary review process. In our case, this could be done by inspecting them manually using the threshold given above, and thus we're ensuring that 298 tweets are labeled correctly by looking up 15 tweets that occur at least t times. However, we decided to not do it as it did not seem to be the goal of this project and would bias the leaderboard results comparable to the other teams.

3.1.2 Data Pre-processing & Engineering. We started by removing all the duplicate tweets from the training dataset. This yielded a new train dataset of 2270482 samples. We used two different dataset formats depending on whether we needed to work with Hugging-Face pre-trained models or PyTorch. This allowed us to improve the speed and memory performances of our programs. We also had to resort to different tricks to be able to load relatively big models and datasets on the GPU; we further discuss this issue in the Discussion section.
For some specific pre-trained Transformer models, it was required to modify some of the tokens as the pre-trained models would recognize them better. For example, for fine-tuning deBERTa, we converted "<user>" to "@user" and "<url>" to "http". For their fine-tuning, we used the Huggingface Transformers library [17] to load the appropriate tokenizer for each model. It is crucial to tokenize the dataset in the same way as the data the model was pre-trained with, otherwise, the results are significantly worse. Regarding the data processing used for baseline models, we can refer to the Baseline Models section.

3.2 Evaluation and baselines

One of the main goals is to obtain a high accuracy on the (private) test set, we necessarily have to define a representative and robust evaluation procedure for our models. In a practical setting, a huge increase in resources for only a minor increase (e.g, +0.001) in accuracy would not be justifiable in most cases. We kept this fact in mind while working on this project and started by working with models that could run faster than others and could be scaled up to much higher capacity, and complexity if necessary.

Considering the size (i.e., $10'0000$ samples) of the test set, we chose to use a train/validation split that would yield a validation set of around $30'000$ samples. This allowed us to keep a considerable amount of tweets for training while having a validation set - whose ideal goal is to mimic the test set - close by a factor of 2 to 5 to the test set's size in order to have a better insight of our model's current generalization ability.

The main metrics observed were the error loss and the accuracy. Others, useful to monitor the program's speed performance for a run, were the total number of steps to perform, the time taken for k (e.g., $k = 1000$) steps, the number of steps per second, and the number of samples seen per second. The values of the aforementioned metrics were displayed numerically directly in the console as well as visualized with line plots through WandB[1]. The time took for a model to perform a complete run, the number of samples seen, the steadiness and smoothness of the learning process, etc., were important information to decide on what model to run, with which hyper-parameters, with what data, etc. Naturally, before comparing all the metrics between model runs, we made sure that the runs could indeed be comparable. In particular, we would use the exact same (fixed but extracted randomly from the randomly shuffled train set) validation set for the run being compared. A seed was fixed and used for all the randomness in the code.

We decided to not perform any kind of grid search to find the best suitable hyper-parameters. The reasons were the following. The baseline models (see 3.2.1) were many (i.e., more than five) and had comparable (but not competitive) performances between them after a bit of manual hyper-parameter tuning. Of course, that is noting that a model such as a XGBoost classifier will be more tunable than a Naive Bayes classifier, but will also be more time-consuming to tune. The pre-trained Transformer models simply and quickly outperformed the more simple models, hence becoming our main focus for the rest of the project competition and effectively attributing the "baseline model" title to the other simpler models (i.e., not Transformer-based). This is not surprising as the *SOTA* for general NLP tasks as well as sentiment analysis is greatly dominated[2] by Transformer-based models (see for example the Related Work section). In the Models section, we describe in great detail the models we mainly worked with and on, as well as the key procedure of fine-tuning relatively large pre-trained Transformer models.

3.2.1 Baseline models. Regarding the baseline models, we tried multiple approaches for the preprocessing and for the models. We noticed that the preprocessing strongly affects our models' performance. So for the preprocessing we tried quite a few approaches after looking for known vectorizing techniques. The Scikit-learn library [18] implements a few interesting vectorizing techniques: HashingVectorizer, TfidfVectorizer, CountVectorizer. We also explored some other techniques of tokenization, and even of combining these Scikit-learn modules with other libraries, but ended up only using the first two techniques as they provided us with the best results. (Also the CountVectorizer is redundant as TfidfVectorizer uses it internally). We also decided to try using a vectorizer from the transformers that performed great, but this leads to poor results since it results in quite sparse matrices which perform poorly on simpler models.

With regard to the chosen models, we decided to opt for simple models such as a Naive Bayes classifier, Linear Support Vector Classification, Logistic Regression Classifier, Random Forest Classifier and an XGBoost classifier. The linear models performed the best, Logistic Regression with the TfidfVectorizer scoring 0.83 accuracy and the Random Forest Classifier with the same preprocessing performed the worst at 0.71 accuracy.

4 MODELS

All state-of-the-art NLP models are transformer-based. Nowadays, it is impossible to achieve good performance on NLP Tasks without using transformers as a backbone. Further, sentiment classification is a well-studied field and numerous transformer models are designed and pre-trained for this task. Therefore, in order to achieve a high score on the leader board, we decided to use pre-trained transformer model as the backbone. Further, we also implemented

[1]https://wandb.ai

[2]https://gluebenchmark.com/leaderboard

custom transformer models using transfer learning and regularization techniques for learning more robust feature mapping. All the pre-trained and custom models are then fine-tuned with the given training dataset.

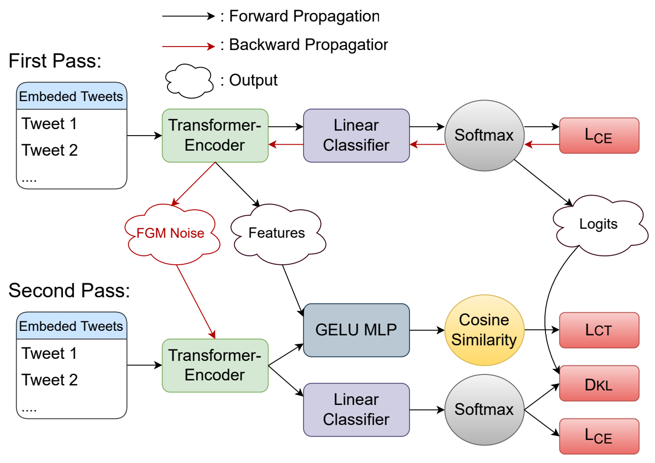

Figure 1: The proposed Contrastive Adversarial Model with KL-Divergence for tweets sentiment classification. L_{CT} is the contrastive loss, L_{CE} is the cross-entropy loss, and D_{KL} is the KL-Divergence.

4.1 CA-KL Model

In this subsection we describe our proposed model for solving the tweets' sentiment classification problem: the contrastive adversarial model with kL-divergence, in short, CA-KL Model. The model combines the idea of transfer learning, contrastive learning, and Bayesian regularization of neural networks and is inspired by the SCAL model in the paper[19], which proposed a learning framework for using contrastive learning in NLP task. Contrastive learning was originally proposed to solve unsupervised computer vision problems[20]. The idea is to create a robust representation of the images such that the representation of an image with slight perturbation should be similar to the representation of the original image and different from representations of other images. In sentiment classification, if one views the deep neural network approach as the composition of a deep neural feature extractor and a classifier head above, it would be natural to adapt the contrastive learning framework into the feature extractor part so that a more robust deep neural feature extractor can be learned. However, unlike in computer vision, a perturbation for natural language sentences can be highly non-trivial and difficult to define. A natural approach here would be to use the adversarial learning [21] in order to generate meaningful perturbations. To this end, there are two common approaches to generating adversarial noises: the fast gradient sign method (fgsm) and fast gradient method (fgm) [22]. In our approach, we chose to use the fast gradient method. However, there is no specific reason against using fast gradient sign method. Due to time shortage, we did not have time to exploit all hyperparameter combinations as well as the choice of adversarial method. For the similarity metric for computing similarities between the representations, cosine similarity is used. This is a standard choice for representations as numeric vectors.

The aim of applying contrastive learning is to create a more robust feature mapping that should not be much influenced by small perturbations. To this end, we added a KL-Divergence loss for minimizing the prediction distribution differences between original tweets and tweets with perturbation.

The CA-KL Model uses a transformer-based Deberta encoder [23] as the backbone and applies transform learning based on that. A full pass of the model consists of two sub-passes, overall two forward propagation, and one backward propagation. As shown in the figure 1, the first sub-pass consists of one forward propagation and one backward propagation. The forward pass computes the features produced by the encoder, the logits of predictions, and the cross-entropy loss computed from the logits. Further, the backward pass computes the gradient of the embedded tweets in order to generate adversarial noises. The second sub-pass takes as input the embedded tweets and adds the adversarial noises generated by FGM to the embeddings. It then computes the contrastive loss, cross-entropy loss, and the KL-Divergence loss using one forward pass. The contrastive loss is defined as:

$$L_{CT} = -\Sigma_i \log softmax_i(sim(x_i, x^p)/T) \tag{1}$$

where $sim(\cdot)$ is the cosine similarity, x are the embedded tweets, x^p are the embedded tweets after perturbation and T is the softmax temperature. The idea behind this loss is to maximize the similarity between x_i and the perturbed x_i^p relative to all other irrelevant pairs. The bidirectional KL-Divergence loss is defined as:

$$L_{KL} = \frac{1}{2}(D_{kl}(q||q^p) + D_{kl}(q^p||q)) \tag{2}$$

where q is the logits of the first sub-pass with original tweets, and q^p are the logits of the second sub-pass with the perturbed tweets. Overall the loss L is defined as:

$$L = L_{CE} + \alpha \cdot L_{CE}^p + \beta \cdot L_{CT} + \gamma \cdot L_{KL} \tag{3}$$

where α, β, γ are constant hyperparameters that are task-specific and to be determined by a hyperparameter search.

4.2 BERTweet Model with simplified R-Drop

In this section, we propose a model based on transfer learning and uses a trained regularization technique. The pre-trained model we chose is the BERTweet model proposed in the paper [24], which uses the BERT architecture as a backbone and is pre-trained using the RoBERTa procedure. Since the model is already fine-tuned beforehand, we take the encoder of this model as the backbone, build a multi-layer classifier head onto it, and put several dropout layers in-between, with a relatively high dropout probability. The idea behind this is to implement a simplified version of the R-drop regularization technique[25]. R-drop performs two forward passes in each training iteration, ensemble both outputs and use a KL-Divergence term to penalize the distance between the two predictions because they can be viewed as two independent samples from the predictive distribution [26]. However, running a big model twice takes a lot of time. After looking into the original model, we observe that the dropout probability in the encoder is only 0.1, and there are only few dropout layers. Therefore, re-running the complete encoder would not give two significantly different

Model	Test Acc.	Eval. Acc.
BernoulliNB		0.760
LinearSVC		0.820
XGB CLASSIFIER		0.780
RandomForestClassifier		0.710
Logistic Regression		0.830
DeBERTaV3-base	0.907	0.906
BERTweet-base	0.918	0.912
CA-KL	0.904	0.907
BERTweet-base simplified R-drop	0.916	0.913

Table 1: Model Accuracy on validation set/public leaderboard

outputs. So instead, we decide to only ensemble on the classifier head, in which we put several dropout layers with a relatively higher dropout probability. The training is then significantly faster but it still gives the desired regularization.

5 RESULTS

5.1 Environment, Implementation and Training Details

All the custom transformer models we proposed are implemented in PyTorch[27] and the training is done using the HuggingFace[28] hub and packages. Also, all pre-trained Transformers were loaded from the HuggingFace hub. The custom transformer models are trained on ETH Euler Cluster with one NVIDIA GeForce RTX 2080 Ti. The baseline models (see 3.2.1) such as SVM or Logistic Regression are implemented using sklearn and trained on our local machine with a single GPU. Only the predictions of the test set from the transformer models were submitted to the Kaggle leaderboard because they did not perform sufficiently well on the validation set as well as because of the limited number of submissions.

5.2 Test Result

In this section, we present the results of our proposed models and compare them with the baselines and pre-trained Transformer models. For each model, we report its test accuracy. The test accuracies for baseline models are based on testing the model on a hidden validation subset of the training data. The test accuracies of transformer models are taken from the public score leaderboard. The results are listed in table 1. As we can see, the best baseline model has an accuracy of 0.82, and our proposed transformer models CA-KL and BERTweet with simplified R-drop outperform the baseline models significantly, which can be expected because the transformer backbones are state-of-the-art. BERTweet with simplified R-drop also performs the best on the validation set. However, we also observe that the original pre-trained BERTweet model performs the best on the public Leaderboard. It is because it was pre-trained using 860 million tweets and its hyperparameter settings are already optimal. On the other hand, our custom transformer models potentially require tedious hyperparameter searching for the best performance, which we are unable to do due to the long training time.

6 DISCUSSION

While working on this project, it became evident that to get the highest score on the leaderboard, it is necessary to use a very big pre-trained transformer model and fine-tune it for many hours. Our best results were achieved by fine-tuning a scaled-down version of BERTweet. With more resources, we could surely have fine-tuned the full version of BERTweet and achieved an even higher accuracy. Custom architectures are unlikely to beat huge amounts of data and computing power. Thus, we would expect that even better results would be achievable when training bigger models for a long time on computationally potent hardware. In the context of this project, a little boost in accuracy would not justify the additional monetary expense. However, in some real-world applications, it might be worth it. Finally, we also explored the idea of taking the encoder from our trained Transformers that performed the best and using it as input features to Neural Networks such as a CNN and a LSTM with Attention. Unfortunately, this method did not yield significant results, although it remains an interesting idea to work on.
It would have been possible to use further datasets found on the internet to supplement our dataset. However, we chose against doing this because our dataset is quite special in the sense that many tweets are labeled incorrectly (as described in section 2). Thus, it is likely that additional data would not have led to better results.

7 CONCLUSION

In this report, we presented our approaches to solving the tweet sentiment classification problem. To this end, we have implemented several baseline models logistic regression, XGB Classifier, and random forest classifier with Hashing Vectorizer for feature extraction from raw texts. We observe that they are outperformed by huge transformer-based classifiers which are fine-tuned by training data. We also proposed two novel transfer-learning approaches, CA-KL and simplified R-drop, which take a transformer model as the backbone and regularize the feature mapping using contrastive learning, adversarial learning, and Bayesian techniques. However, due to time shortage and long training time, we can't exploit all hyperparameter combinations, which are crucial for the regularization effect. On the other hand, pre-trained state-of-the-art (SOTA) models are already trained on huge datasets and their hyperparameter combinations are also optimal initially. Therefore, the two novel transformer models did not outperform the pre-trained BERTweet Model, which was already fine-tuned using 860 million tweets. However, we want to emphasize that given enough time to try out hyperparameter combinations, our custom transformer models can potentially perform better and may outperform the existing SOTA model for tweets classification.

REFERENCES

[1] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams engineering journal*, vol. 5, no. 4, pp. 1093–1113, 2014.
[2] B. Liu, "Sentiment analysis and opinion mining," *Synthesis lectures on human language technologies*, vol. 5, no. 1, pp. 1–167, 2012.
[3] E. Cambria, D. Das, S. Bandyopadhyay, and A. Feraco, "Affective computing and sentiment analysis," in *A practical guide to sentiment analysis.* Springer, 2017, pp. 1–10.
[4] K. Denecke and Y. Deng, "Sentiment analysis in medical settings: New opportunities and challenges," *Artificial intelligence in medicine*, vol. 64, no. 1, pp. 17–27,

2015.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[6] I. Mozetič , M. Grčar, and J. Smailović, "Multilingual twitter sentiment classification: The role of human annotators," *PLOS ONE*, vol. 11, no. 5, p. e0155036, may 2016. [Online]. Available: https://doi.org/10.1371%2Fjournal.pone.0155036

[7] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).* Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. [Online]. Available: https://aclanthology.org/D14-1181

[8] K. H. Manguri, R. N. Ramadhan, and P. R. M. Amin, "Twitter sentiment analysis on worldwide covid-19 outbreaks," *Kurdistan Journal of Applied Research*, pp. 54–65, 2020.

[9] A. Moschitti and R. Basili, "Complex linguistic features for text classification: A comprehensive study," vol. 2997, 04 2004, pp. 181–196.

[10] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha, "Ammus : A survey of transformer-based pretrained models in natural language processing," 2021. [Online]. Available: https://arxiv.org/abs/2108.05542

[11] "State of the art sentiment analysis comparison | papers with code," https://paperswithcode.com/sota/sentiment-analysis-on-sst-2-binary, accessed: 2022-07-31.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[13] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[14] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," 2020. [Online]. Available: https://arxiv.org/abs/2006.03654

[15] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," 2019. [Online]. Available: https://arxiv.org/abs/1911.02116

[16] "Twitter search - a smile never goes out of style wear yours every day," https://twitter.com/search?q=A%20smile%20never%20goes%20out%20of%20style%20wear%20yours%20every%20day&src=typed_query&f=top, accessed: 2022-07-31.

[17] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2019. [Online]. Available: https://arxiv.org/abs/1910.03771

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[19] D. Miao, J. Zhang, W. Xie, J. Song, X. Li, L. Jia, and N. Guo, "Simple contrastive representation adversarial learning for nlp tasks," *arXiv preprint arXiv:2111.13301*, 2021.

[20] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive representation learning: A framework and review," *IEEE Access*, vol. 8, pp. 193 907–193 934, 2020.

[21] G. Li, P. Zhu, J. Li, Z. Yang, N. Cao, and Z. Chen, "Security matters: A survey on adversarial machine learning," *arXiv preprint arXiv:1810.07339*, 2018.

[22] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[23] 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.

[24] D. Q. Nguyen, T. Vu, and A. T. Nguyen, "Bertweet: A pre-trained language model for english tweets," *arXiv preprint arXiv:2005.10200*, 2020.

[25] L. Wu, J. Li, Y. Wang, Q. Meng, T. Qin, W. Chen, M. Zhang, T.-Y. Liu *et al.*, "R-drop: Regularized dropout for neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 890–10 905, 2021.

[26] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning.* PMLR, 2016, pp. 1050–1059.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[28] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.* Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6