Building a Robust QA System that Knows When it Doesn't Know

Stanford CS224N Default Project

Diptimay Mishra
Department of Computer Science
Stanford University
diptimay@stanford.edu

Kumar Kallurupalli
Department of Computer Science
Stanford University
kksreddy@stanford.edu

Abstract

Machine Learning models have a hard time knowing when they shouldn't be confident about their output. They falter when they see data that is out of their training distribution. Unfortunately, they falter in a way that the scores don't properly reflect. For example, an MNIST model is capable of predicting a chicken is actually the number 5 with 99% certainty.[1] While models are capable of generalizing to data they haven't seen, they have a hard time knowing what data they can and can't generalize to. A robust QA module should not only be able to do a good job of handling data it wasn't modeled to handle, it should also be able to do a good job of knowing what data it can't handle. The goal of our project is to initially build a good Question Answering model with an architecture that relies on a base of DistilBERT, improve on it through model fine-tuning, better optimization, and then augment the predictions of the model with a confidence score that is high when the model is reliable and low when the model is basically guessing on data it hasn't seen.

(M)

1 Introduction

Q: "How many eyes does a blade of grass have?"
A: "A blade of grass has one eye."

(NLP Model)

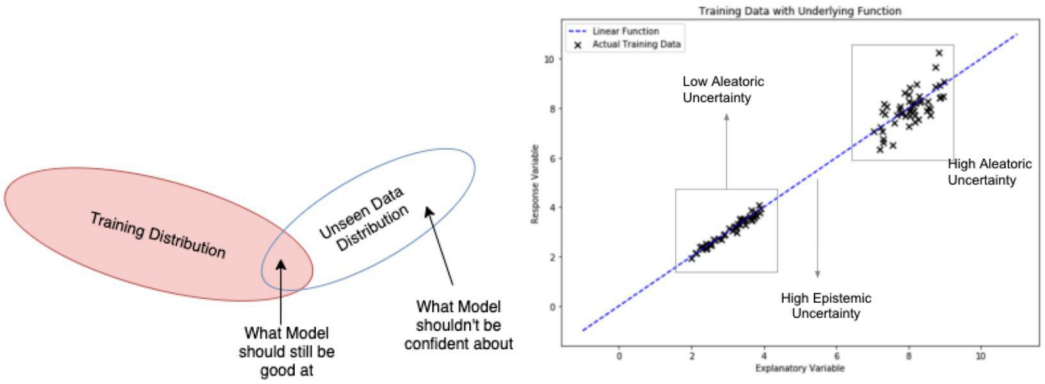
Language Models have shown remarkable performance on many NLP tasks such as classification, question answering, sentence prediction. Question Answering systems aim to take in some context and when asked a question are able to parse the context to reveal the correct answer to the question. Question Answering in particular is wildly applicable area given it's ability to parse entities and relationships.

However, as seen in the above example, Neural Net NLP models falter sometimes when they are exposed to data that is foreign to them. They confidently predict output that ends up being wrong and the complexity and the ability to learn so well from training data that makes Neural Networks so effective also makes them prone to this behavior.

2 Related Work

As a part of our work on this project, we first researched existing approaches in deep learning to improve model performance in this context. We described some of the salient information down below.

[1] https://emiliendupont.github.io/2018/03/14/mnist-chicken/



(a) Data Distributions (b) Baseline - NLL

Figure 1(a): A figure that shows how a LM can be trained on one distribution and potentially be applied on a different one. Keep in mind that if there is no overlap, the model probably won't do well on the unseen distribution. Figure 1 (b): Shows the sources of uncertainty when training a model. When training a robustQA system you are likely to encounter epistemic uncertainty during predictions

2.1 Model Confidence

Of the literature we explored, there was a clear focus on tackling the model confidence problem in the image classification space both Bayesian and Non-Bayesian.

Moon and Kim, 2020[1] work on ordinally ranking data points in the order of confidence. They do this by modifying the loss function to account for which points are being classified correctly in each mini-batch during training and then maximizing the score distribution's skew towards the correct image class using this additional loss term. This way, during inference the final softmax score has the confidence built in based on the separation in probability of the correct class and the wrong classes.

DeVries and Taylor, 2019[2] attempt to add a specific confidence score to the final output layer in addition to modifying the loss function to account for the confidence term. During training the confidence score is used to adjust the model probabilities by multiplying the predicted probability with c and multiplying the correct class with 1 - c.

The benefits of these approaches is that they indirectly make the model better by teaching it not to over-fit to the training data and to generalize better when it does see out-of-distribution data by making it focus on the generic components of language.

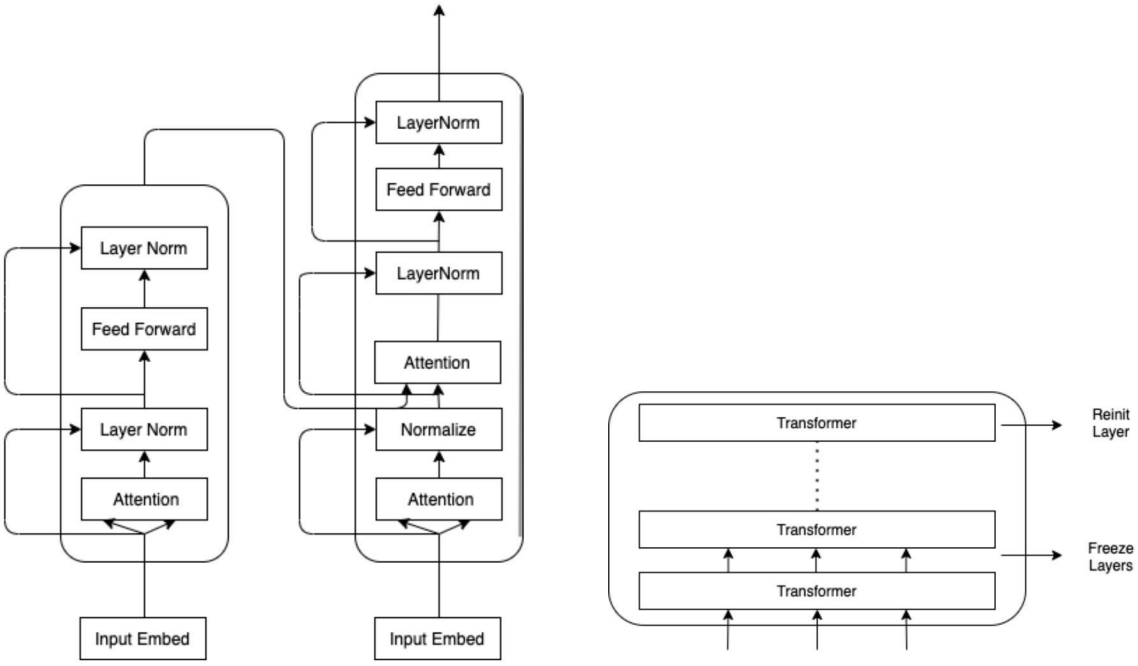
2.2 Model Training and Learning

When using large models like BERT, the key focus is on fine-tuning it to work well on your specific task. We explored some of the literature on how to get more out of your model through fine-tuning and ways to improve the learning process given the typically small amount of data available during this process.

Zhang and Wu, 2020[3] focus on the various hyper-parameters and optimizer decisions that significantly impact model performance after fine-tuning BERT. They look into re-initializing layers, bias correcting the Adam optimizer, increasing training time along with modifications in the architecture like replacing dropout with mixout.

Gururangan and Marasovic, 2020[4] focus on the impact that pre-training the model has on model scores. In addition they specifically look into the impact that pretraining with domain datasets has and show that teaching models to recreate domain data through masking usually leads to better performance. They also analyze the overlap in domains by looking at the unigrams shared between different datasets.

Finally, we aimed to look at the impact of meta learning in neural networks (Hospedales and Antoniou, 2020[5]). The focus of this brand of model tuning is on the improving the optimizer which aims to improve the parameters of the model. The optimizer is shared across all the parameters of the model and over each iteration of model training, the optimizer learns the best settings to minimize loss.



(a) Encoder Decoder Architecture (b) BERT/DistilBERT

Figure 2(a): A figure that shows the architecture of Encoders and Decoders (note: the BERT Pre-Trained Model doesn't use decoders and instead relies on masked tokens) Figure 2 (b): Typical BERT architecture which stacks Transformers

3 Approach

3.1 Preface

Question-Answering requires two capabilities out of the machine learning model: First, the ability to learn important details from a chunk of data and second, the ability to parse a question and retrieve relevant information. Since the introduction of BERT, the use of this transformer based Neural Network model has become a go-to approach for the QA task. These models use the base weights from BERT and then fine-tune on specific datasets. Unfortunately, while they outperform models trained from scratch on the source data, they fail when extended to out-of-domain data[2]. The failure of BERT based models in being robust to changes in datasets and not being capable of knowing when they are unreliable presents a problem that needs to be tackled. For our project, our goal is to build a more robust DistilBERT based model and then to augment it to recognize the cases where it isn't robust enough. Our approach over the course of this project was forked in two directions. The first focused on fine-tuning the model itself through approaches like transfer learning, training for longer epochs, mix-out and re-initializing layers. The second focuses on augmenting the model by providing a confidence score to enhance the model's reliability in real world usage.

3.2 Model Pre-Training

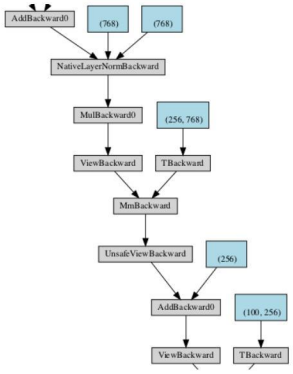
To make the DistilBERT model more affine to our in-domain datasets, we resorted to unsupervised pre-training on the datasets we were provided. We did this by first concatenating all the contexts in the various datasets into one large file and then treating this larger dataset as a big chunk that we can manipulate for this purpose.

To pre-train we relied on corrupting the data. We selected random segments of text from our now larger corpus and utilized a span corruption function to replace a sequence of characters with unique tokens

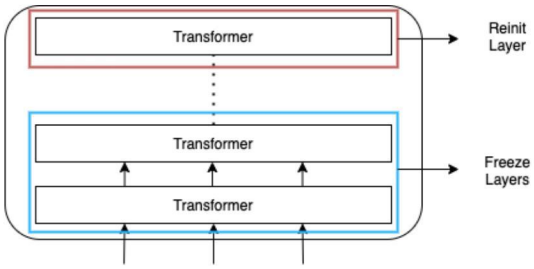
[2]https://www.kdnuggets.com/2020/03/bert-fails-commercial-environments.html

3.3 Model Tuning

Fine tuning pre-trained language models like BERT is the most popular approach to get better results in NLP tasks. This saves a lot of time and resources in training models from scratch. In addition, usage of pre-trained self supervised models in NLP out performs models that are trained from scratch. However, there are known limitations to the performance of BERT on smaller datasets. The first one is that BERT/DistilBERT are trained on a variety of tasks which makes it a model that is easier to generalize but on the flip side, it needs to be fine-tuned on the small data to do better at this specific task. The second is that the finetuning process is susceptible to the distribution of data in the smaller datasets. It might be overly influenced by one dataset over the other based on data size.



(a) Additional Linear Layers



(b) Freeze and Re-init

Figure 3(a): A figure that shows how we extended the model architecture by adding additional linear layers
Figure 3 (b): We froze the initial layers to reduce training time and re-init the later layers to improve performance

We aim to improve on this by:

1. Training for longer epochs: Generally the DistilBERT model is fine-tuned for 3 epochs. We decided to train for longer to see the impact it would have on the loss
2. Freezing all but the last layers of the BERT model: To reduce the training time, we froze the initial layers of DistilBERT which handle semantic clustering[3] as seen in Figure 3(b)
3. Re-initializing the weights: The later layers of DistilBERT are not optimized for the task at hand since they usually collate entities with attributes[3]. So we replaced the later layers with the appropriate re-initialization for the given type as seen in Figure 3(b)
4. Using Mixout: Mixout is a form of dropout that instead of dropping the value to 0, uses the pre-trained value to prevent catastrophic forgetting. We replaced the dropout layers in the transformation with mixout layers that had a similar probability of dropping/replacing the node with nothing/older values that are forgotten.
5. Train with bias correction Bias correction is used to negate the impact of the initial selection of m_0. Since the momentum and scaling from Adam are essentially a weighted mean of values, the initial selection that guides these means matters and bias correction aims to correct how much those initial selections matter.
6. Train with additional layers We also attempted to add additional linear layers to try improve the model's ability to fit complex patterns.

3.4 Model Confidence

We modify the network to add a confidence measure. The confidence measure is used to affect the final probability scores by multiplying the confidence with the probability score for a class and then adding it to (1-c) times the corresponding class label (See Equation (1)).

$$p_i' = c \cdot p_i + (1 - c)y_i \tag{1}$$

In the case of a QA model, we now have two probability distributions instead of one. So instead of applying our new c term on one set of probabilities we do it for both leading to the following equation

$$L = -log(p_{start}(i)) - log(p_{end}(j)) \tag{2}$$

We then modify the default task loss (See Equation (2)) by then adding this confidence loss to it (Equation (2)) where lambda is a regularization hyper-parameter.

$$L = -log(p_{start}(i)) - log(p_{end}(j)) - \lambda log(c) \tag{3}$$

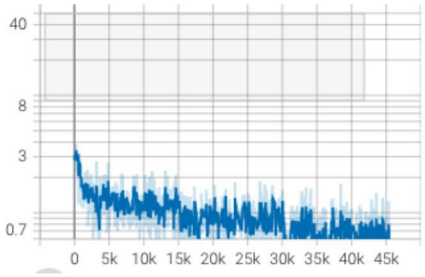
4 Experiments

4.1 Details

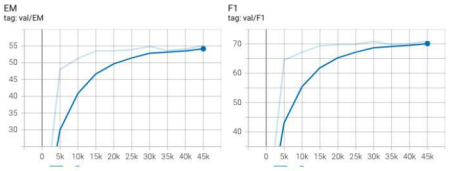
Given the time constraint, we decided to first check if the model tuning methods listed in the section above would provide an improvement to the base DistilBertForQuestionAnswering model.

4.2 Baseline

To establish a baseline, we trained the DistilBertForQuestionAnswering model with bias correction turned off.
BaseLine Results: Training Time =. 15 hour 40 min EM = 55.07 F1 = 70.95



(a) Baseline - NLL



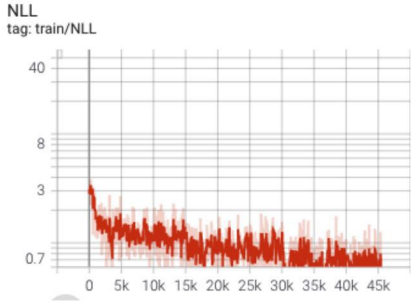
(b) Baseline - performance

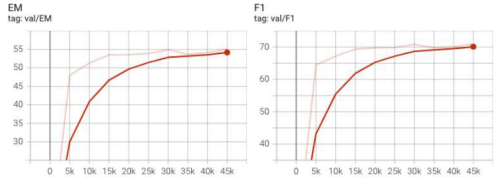
4.3 Bias Correction

We then added the bias correction steps (see Equation(4)) to the ADAM optimization. It turned out there wasn't any difference in the resultant accuracy although the training time was slightly on the higher side.

$$\hat{m}_t \leftarrow m_t/(1-\beta_1^t), \hat{v}_t \leftarrow v_t/(1-\beta_2^t) \tag{4}$$

Training Time =15 hr 52 min(3 epochs) EM = 55.07 F1 = 70.95





4.4 Freezing the initial layers

We tried to measure the impact of freezing the initial layers of DistilBERT on the performance.There isn't a significant improvement with the model evaluation performance, however, it does speed of the training of the model.

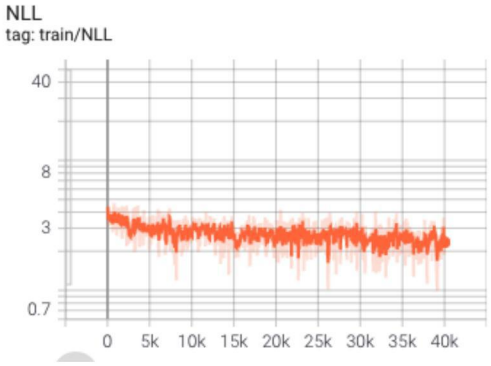


Figure 5: Freeze First 4 layers of DistilBERT

4.5 Re-initializing BERT pre-trained layers

As part of applying transfer learning to fine tuning using BERT, the common practice is to initialize all the layers with pre-trained weights except the output layer. This experiment is to verify the impact of using the complete layer which is our baseline verses re-initializing the higher layers that are closer to the output. This is motivated by the intuition t We attempted to reinitialize the pooling layers and the top L $\in N\ BERT\ transformer\ blocks.$
$andL \in 2, 4, 6$

We observe that re-initializing the lower layers have a positive impact on the performance of the model. This observation is inline with the intuition that hat lower layers learn more general features while higher layers learn that are closer to output learn more specific features.

For 2 layers,
Training Time =12 hrs 31 min EM = 55.72 F1 = 71.8
For 4 layers,
Training Time =10 hr 14 min (2 epochs) EM = 54.96 F1 = 71.51

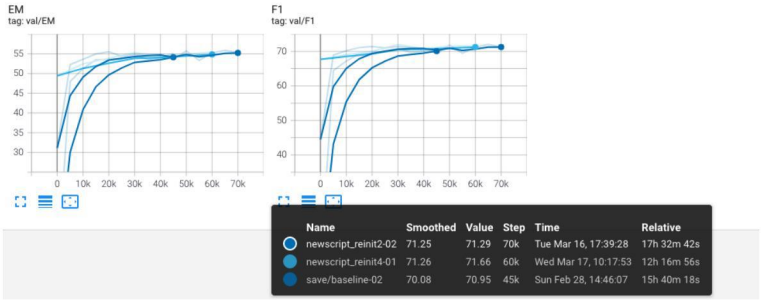


Figure 6: Re-initializing BERT pre-trained layers

To be tested: Sensitivity to number of layers re-initialized: In future work, we intend to find out the threshold of the number of layers after which re-initializing starts to negatively hurt the performance.

4.6 Regularization using mixout

Mixout is a regularization technique to address the instability in BERT fine tuning process. This is achieved by by constraining the model to stay close to the pre-trained weights rather than deviating

away from them. In our experiment, for each training iteration - we replace the the model parameters with the pre-trained weights with probability $\mathcal{P}$,
where $\mathcal{P} \in 0.3, 0.5, 0.9$

We observe an increase in performance with the mixout approach. A higher probability score provides a higher stability and performance to the model without losing out on the training time

For mixout with probability 0.3,
Training Time =14 hr 24 min(3 pochs) EM = 55.56 F1 = 71.7
For mixout with probability 0.5,
Training Time =15 hr 18 min(3 pochs) EM = 58.06 F1 = 73.93

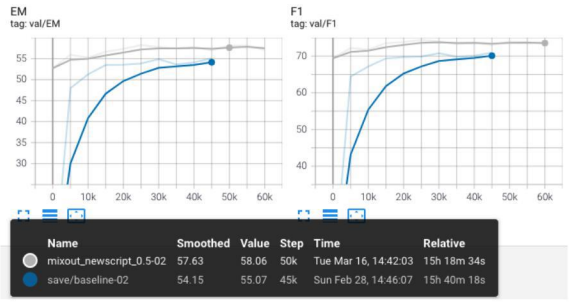


Figure 7: Regularization using Mixout - Performance

We tried to measure the impact of freezing the initial layers of DistilBERT on the performance.There isn't a significant improvement with the model evaluation performance, however, it does speed of the training of the model. We then tried to measure the impact of freezing the first 4 layers of DistilBERT, re-initializing the last 2 Transformer layers and adding Mixout to the last linear layer. Unfortunately, due to the time constraint, we had to run all experiments in the same trial. Figure 3 has the Negative Log Likelihood plotted for this experiment and in a contract to the previous two runs, we can see that the performance is much worse and the loss plateaus sooner.

Running all these changes simultaneously means we couldn't isolate what caused the poor performance. A possible cause could be due to too many layers being frozen which meant the loss could never converge and kept oscillating after it plateaued.

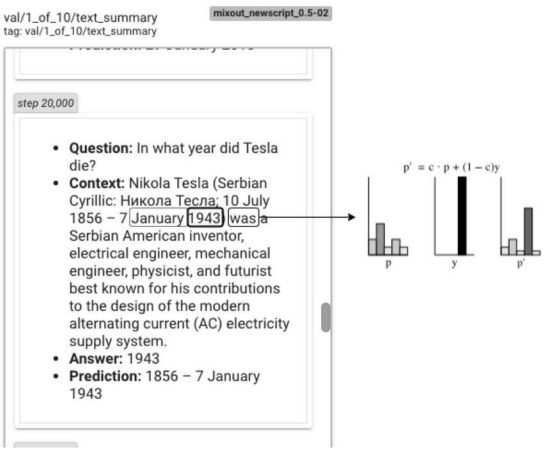
Issues encountered: Applying mixout technique came with the disadvantage of being memory intensive. This led to crashing of our training server. We mitigated it by reducing the batch size.

4.7 Experiment Result Summary

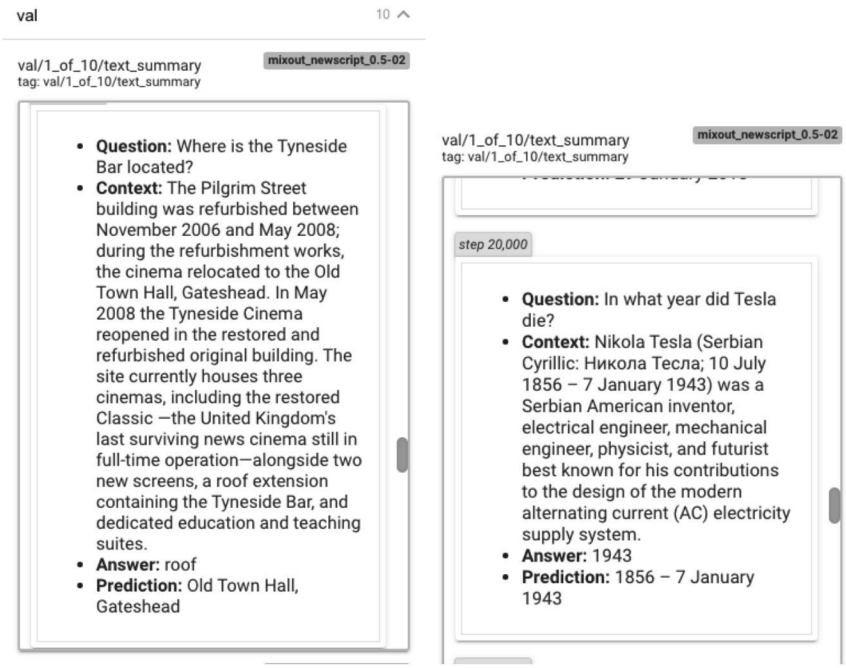
	EM	F1	Training Time
Baseline	55.07	70.95	15 hr 40 min(3 epochs)
Bias Term	55.07	70.95	15 hr 52 min(3 epochs)
Reinit 2 layers	55.72	71.8	12 hrs 31 min
Reinit 4 layers	54.96	71.51	10 hr 14 min (2 epochs)
mixout prop 0.3	55.56	71.7	14 hr 24 min(3 pochs)
mixout prop 0.5	58.06	73.93	15 hr 18 min(3 pochs)

4.8 Confidence

We added the additional confidence output node to our model and attempted to evaluate the effect it has on model training. Unfortunately our model repeatedly crashed during training and by the time we reduced our batch size we didn't have enough time to fully train this variation of the model.



4.9 Qualitative Analysis



(a) sample 1 (b) sample 2

Sample 1: This is a sample prediction from our experiment using mixout generalization technique. We can see that model wrongly predicted the location of Tyneside Bar. The prediction "roof" was probably influenced by the word "containing". Also, the model is not able to find the relationship between the location that is described at the beginning of the context with the Tyneside Bar which is mentioned towards the end of the context.

Sample 2: In this case, the model is unable to interpret the meaning of the dates that are laid out after the name. Since, there is no mention of the relevance of the dates, the model is unable to predict the birth and death dates. Most likely, the training corpus doesn't have enough examples that explains the interpretation of dates that features right after the name that represents an individual's life span.

5 Future Work

As mentioned above, the addition of our confidence term to the model and to the loss has resulted in worse performance by the model. We would like to complete the training with smaller batch size

and observe the impact on the accuracy fo the model. Also, we would like to further experiment on the sensitivity of the model to the number of layers that were re-initialized as compared to using pre-trained weights. Finally, Our future goal with this project is to work on improving the fine-tuning through the implementation of meta-learn and improving the logic of our confidence metric for this use-case.

References

[1] Jooyoung Moon, Jihyo Kim, Younghak Shin, and Sangheum Hwang. Confidence-aware learning for deep neural networks, 2020.

[2] Terrance DeVries and Graham W. Taylor. Learning confidence for out-of-distribution detection in neural networks, 2018.

[3] Arzoo Katiyar Kilian Q. Weinberger Yoav Artzi Tianyi Zhang, Felix Wu. Revisiting few-sample bert fine-tuning, 2020.

[4] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks, 2020.

[5] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey, 2020.