PDL Challenge 2: Enhancing Reliability in Deep Neural Networks: Dropout, Sparsification, and Ensemble Techniques using VGG16 for CIFAR-10

SR+ITL

April 9, 2024

Abstract

Deep Neural Networks (DNNs) have demonstrated exceptional capabilities in various domains. However, a pervasive challenge in deploying DNNs is their tendency towards overconfidence in predictions, which can obscure the reliability of their outputs and complicate decision-making processes in critical applications. In machine learning, ensemble techniques have traditionally been used to to mitigate overconfidence by leveraging the collective intelligence of multiple models. The practical of ensembling techniques in deep learning results in increased computational complexity. This challenge explores the balance between enhancing model reliability through ensembles and maintaining computational efficiency via model sparsification, using the CIFAR-10 dataset as the experimental ground. By modifying the VGG16 architecture and implementing dropout, sparsification, and ensemble strategies, we aim to dissect the dynamics of DNN overconfidence and propose a methodology that not only addresses the issue but also paves the way for creating more dependable and efficient DNNs for image classification and beyond.

1 Problem Setting

Certainly, we can create a subsection that delves into the concept of Kullback-Leibler (KL) Divergence as a means to measure the distance between two probability distributions. This will set the groundwork for utilizing KL Divergence to evaluate the alignment between a model's confidence in its predictions and the empirical distribution of prediction errors, ultimately aiming for a model whose confidence accurately reflects its actual performance.

1.1 Understanding Kullback-Leibler (KL) Divergence

A fundamental aspect of assessing the reliability of a Deep Neural Network (DNN) is the ability to quantitatively measure the alignment between the model's confidence in its predictions and the actual accuracy of these predictions. The Kullback-Leibler (KL) Divergence provides a powerful mathematical framework for this purpose, offering insights into how closely the model's perceived reliability (expressed through its confidence levels) matches the empirical distribution of its errors.

The Kullback-Leibler Divergence, a concept from information theory, quantifies the difference between two probability distributions, P and Q, over the same variable X. For discrete variables, it is defined as:

$$D_{KL}(P \parallel Q) = \sum_{x\in} P(x) \log\left(\frac{P(x)}{Q(x)}\right). \tag{1}$$

where P and Q are two probability distributions defined over the same alphabet $\mathcal{X}$.

The conditional version of the KL divergence is defined as

$$D_{KL}(P(Y|X) \parallel Q(Y|X)) = \sum_{x\in X} P(x) \sum_{y\in Y} P(y|x) \log\left(\frac{P(y|x)}{Q(y|x)}\right). \tag{2}$$

and it expressed the KL divergence of the random variable Y when conditioned over the random variable X.

1.2 Application to Model Reliability

For a DNN to be considered reliable, its confidence in making correct predictions should closely mirror the actual outcomes. That is, when a model assigns high confidence to a prediction, the likelihood of that prediction being correct should be correspondingly high. Conversely, a misalignment between model confidence and accuracy suggests overconfidence or underconfidence, both of which are undesirable traits in critical decision-making processes.

To utilize KL Divergence in this context, we propose the following approach: Let us consider a DNN used for classification and consider the softmax output as a measure of reliability for the decision. To estimate of this measure of reliability by measuring the KL divergence between the following two distributions

1. **Empirica Distribution of the Prediction**– $P_{Y|X}$**:** After the model makes predictions on a test set, we evaluate the empirical distribution of the label prediction as follows: $P(j|i)$ is the probability that an input from class i is assigned by the model to class j
2. **Model Confidence Distribution** – $Q_{Y|X}$**:** Concurrently, we aggregate the model's confidence levels for the classes, deriving a distribution that reflects how the model's confidence across the various classes. Accordingly, $Q(j|i)$ is the average of the softmax output for the input in class i for the output label j.
3. **Calculating Conditional KL Divergence:** Given the test set, we calculate the KL divergence and the conditional KL divergence using (1) and (2). Note that (1) is applied for each class, so that we obtain a distance measure for each true label. Also, note that for (2) $P(X)$ is the probability of a label so that we average the KL over the true label distribution.

   Overall, a lower value of KL Divergence indicates a closer match between the model's confidence and its actual performance, signifying higher reliability.

2 Challenge Description

This challenge proposes an exploration of ensemble methods to enhance the reliability of deep neural network predictions, with a specific focus on a modified VGG16 architecture tailored for the CIFAR-10 dataset. CIFAR-10, consisting of 60,000 32x32 color images across 10 classes, presents a diverse and challenging benchmark for image classification tasks. Our objective is to investigate how variations in the network's architecture, particularly in the integration and manipulation of dense layers, can affect model performance and reliability.

2.1 Modifying VGG16 for CIFAR-10

The foundation of our experiment is a version of the VGG16 model, a convolutional neural network originally designed for large-scale image processing. We adapt VGG16 for CIFAR-10 by appending three dense layers at the end of its convolutional base. These dense layers are crucial for our ensemble creation strategies, serving as the focus of our modifications for reliability enhancement. The convolutional layers of the VGG16 model will be fixed to preserve the feature extraction capabilities of the network, ensuring that our ensemble strategies primarily leverage and modify the model's decision-making process.

2.2 Creating a Model Ensemble

Our ensemble creation takes two distinct approaches, both centered around the manipulation of connections within the dense layers of the modified VGG16 architecture:

- **Task 1: Dropout-Induced Random Edges** In the first approach, we employ dropout as a means to randomly disable connections between neurons in the dense layers during training.

This method introduces randomness into the network, simulating an ensemble of sub-models with varying architectures for each training iteration. The primary goal is to investigate how this induced variability affects the overall ensemble's ability to generalize and predict more reliably, comparing the collective performance against that of a single, unmodified network.

- **Task 2: Sparsification of Dense Layers**

  The second approach explores the concept of sparsification, where connections between neurons in the dense layers are selectively pruned to create a more streamlined and efficient model. This technique reduces the complexity of the network, potentially enhancing computational efficiency and performance by focusing on the most informative connections. Similar to the dropout approach, we aim to create an ensemble of sparsified models to assess the impact on performance and reliability compared to the baseline VGG16 model.

3 Creating a Baseline Model

Consider the following model in tensorflow/keras inspired by VGG16.

```
def create_modified_vgg16():
    # Load the VGG16 model, pre-trained on ImageNet, without the top layer
    base_model = tf.keras.applications.VGG16(include_top=False, input_shape=(32, 32, 3), weights=None, pooling='avg')

    # Freeze the layers of the base model
    for layer in base_model.layers:
        layer.trainable = False

    # Create the model
    model = models.Sequential()

    # Add the base model
    model.add(base_model)

    # Flatten the output layer to 1D
    model.add(layers.Flatten())

    # Add new dense layers
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dense(256, activation='relu'))
    # Output layer with softmax activation
    model.add(layers.Dense(10, activation='softmax'))

    return model

# Create and compile the model
modified_vgg16 = create_modified_vgg16()
modified_vgg16.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Summary of the model
modified_vgg16.summary()
```

Start by training the model until you reach a global accuracy of around 70%. Note that you should only train the last three dense layers

In the following, we shall create 5 networks for each task. These five networks will contain on average 20% of the connections between the three dense layers. We will then try to improve the performance of the network in terms of accuracy and reliability, as we discuss in the next section.

3.1 Plot accuracy vs reliability

Our objective here is to verify the interplay between accuracy and reliability during the training process. In particular we wish to observe that the network is generally overconfident, outputting a high reliability even when its accuracy is low.

In particular, to calculate the reliability you should use KL divergence between the predicted class probabilities and the empirical distribution of the class labels given the input. This is obtained by

extracting the softmax outputs of the network to compare against the true labels. Conceptually, give a set of weights and a test set do the following:

1. Obtain the empirical distribution of the model prediction per each class. This is obtained by taking one class in the test set and calculating the empirical distribution of the predictions over this label. For each label, this is a vector of ten elements – the ten labels– which sums to one.
2. Stack the empirical distribution for each class into a matrix of 10×10. Let the row be the true label and the column the predicted label. It position (i, j) of this matrix you have the element $P_{j|i}$ as described in Sec. 1.2.
3. Obtain the average reliability per each class. This is obtained by averaging the model prediction, i.e. softmax, over a certain label. For each label, this is a vector of ten elements which subs to one.
4. Stack the average reliability for each class into a matrix of 10×10 elements. Let the row be the true label and the column the predicted label. It position (i, j) of this matrix you have the element $Q_{j|i}$ as described in Sec. 1.2.
5. Calculate the conditional KL divergence. This is a single scalar value.

Next, for the training process described in Sec. 3 plot accuracy and reliability and comment on their relationship

4 Task 1: Dropout-Induced Random Edges

Having understood the relationship between accuracy and reliability in the baseline model, we can then proceed in creating a model ensemble using dropout.

- Fix the convolutional layers of the architecture in Sec. 3
- Create five versions of the baseline model in which where 80% nodes of the last three layers are randomly eliminated. This is equivalent to adding a dropout layer to the network. After the nodes have been eliminated, retrain the five networks.
- At inference, the five networks act as a model ensemble: the input is presented to all five models and the output is obtained as the average of the softmax output of each of the five models
- Evaluate the accuracy and the reliability of the model examples as in Sec. 3.1
- Comment on the overall performance of the model ensemble, for example in terms of over-fitting.

Note that the dropout probability is chosen so that the overall computational complexity between the baseline model and the model with dropout is roughly the same.

Next, experiment with the network structure and training process so as to improve the network reliability

- **Add the KL to the loss function**

  Integrating the Kullback-Leibler (KL) divergence into the loss function represents a strategic enhancement aimed at directly addressing and improving model reliability. The challenge lies in balancing the contribution of the KL divergence to avoid overshadowing the primary loss component, thereby ensuring that model training remains focused on achieving high accuracy while enhancing the reliability of its confidence estimates.

- **Introduce Temperature Scaling in Soft-max:** Temperature scaling modifies the soft-max layer to produce more calibrated probabilities. It involves dividing the logits by a temperature parameter before applying soft-max. A temperature greater than 1 softens the probabilities, potentially leading to more reliable confidence estimates.

- **Regularization Techniques:** Implement or adjust regularization techniques, including L1/L2 regularization on the dense layers, to prevent over-fitting and encourage the model to learn more generalized representations, potentially improving reliability.

- **Differ Ensemble Methods:** Leverage ensemble techniques to improve reliability. Training multiple versions of the model with slight variations and then averaging their predictions can lead to a more reliable prediction that better represents uncertainty.
- **Introduce extra outputs to represent confidence levels** To implement this, one might augment the model with additional dense layers dedicated to confidence estimation, branching out from either the final layers of the base model or at intermediate points within the architecture. These layers could output a single confidence score per prediction or a vector of confidence scores across classes.

5 Task 2: Sparsification of Dense Layers

In this part, we create a model by eliminating edges, instead of nodes.

- Fix the convolutional layers of the architecture in Sec. 3
- Create five versions of the baseline model in which where 80% edges of the last three layers are randomly eliminated. This is equivalent to random sparsification. After sparsification, make sure to retrain the model.
- At inference, the five networks act as a model ensemble: the input is presented to all five models and the output is obtained as the average of the softmax output of each of the five models
- Evaluate the accuracy and the reliability of the model examples as in Sec. 3.1
- Comment on the overall performance of the model ensemble, for example in terms of over-fitting

Next, experiment with the network structure and training process so as to improve the network reliability as you did in Task 1.