Classification on CIFAR with a Bayesian Network

In this project Bayesian neural network is used to solve image classification task.

In this implementation, I have built a Bayesian Convolutional Neural Network (CNN) for the task of image classification. At the first stage of the project the Bayesian NN implementations is trained on CIFAR-10, after succeeding a harder dataset will be used. For CIFAR-10, the goal is to classify images into one of 10 categories, such as "cat," "dog," "airplane," etc., using a network that not only makes predictions but also measures how uncertain it is about those predictions.

Key Elements of the Model:

1. CIFAR-10 Dataset:
    - The CIFAR-10 dataset consists of 60,000 images, divided into 10 different classes, with 6,000 images per class. The images are small (32x32 pixels) and are in color (RGB). The task is to train a model that can correctly classify these images.

What I Have Done in the `BayesianCNN` Implementation

In this implementation, I created a Convolutional Neural Network (CNN) that uses **Bayesian methods** to handle uncertainty in its predictions. Here's a breakdown of what I did:

1. **Bayesian Convolutional Layers (`BayesianConv2dFlipout`)**:
    - I implemented custom convolutional layers where the weights (and biases) are treated as **random variables** with a distribution (using a **Normal distribution**).
    - These layers use a technique called **Flipout**, which helps in efficiently approximating the uncertainty in the model's weights during training.
    - I defined the **mean** and **log-standard deviation** for each weight and bias. The **mean** represents the most likely value, and the **log-standard deviation** tells us how much uncertainty is associated with that weight.
2. **Bayesian Fully Connected Layers (`BayesianDenseFlipout`)**:
    - I created fully connected layers similar to the convolutional layers, where weights and biases are also probabilistic.
    - These layers follow the same principle: weights and biases are learned as distributions, not as fixed values.
3. **Bayesian CNN Model (`BayesianCNN`)**:
    - The `BayesianCNN` class connects the **Bayesian convolutional layers** and **fully connected layers** into a complete neural network.
    - The network architecture is similar to a regular CNN with several convolutional layers for feature extraction followed by fully connected layers for classification.
    - I used the **Flipout** technique for efficient uncertainty estimation during the forward pass of the model.
4. **Forward Pass with Uncertainty**:

- In the forward pass, I used both **stochastic (uncertain)** and **deterministic (fixed)** modes for sampling the weights and biases.
- This allows the model to make predictions while also estimating how uncertain it is about those predictions.
- The final output is a **log-softmax** result for classification, which gives the model's predicted class probabilities.

5. **Uncertainty in Predictions**:
    - By using these Bayesian methods, the model can not only make predictions but also tell us how confident it is about those predictions.
    - This is helpful in situations where we want to know whether the model is certain or uncertain about its output (e.g., for tasks involving out-of-distribution detection).

The model was only achieved around **10% accuracy**, which is quite low. This result suggests that the model architecture may not yet be fully learning the patterns in the data.

In the future, I plan to continue training the model for more than **100 epochs** to see if it can improve its performance. This will give the model more time to learn from the data and I will be able to derive conclusions about the architecture's ability to learn. Along with this, I'll keep monitoring the model's uncertainty estimates to better understand how confident it is in its predictions.