In order to solve this task, we decided to implement the dropout neural network. As explained in class and in the related paper, this network can thought as performing (approximate) Bayesian inference because at prediction time, the predictive distribution is calculated as an average (i.e. approximation of expectation) of the predictive distributions which are given by randomly selecting to set weights to 0 or not. This differentiates this approach from the classical dropout network, where dropout is performed at training time only.

In our implementation, we first of all modified the DropoutTrainer class, where we wrote the loss function (which in this case is the CrossEntropy plus the L2 regularization term) and the predict\_probabilities module, where we performed a MonteCarlo sampling (with n\_sample=100) from our trained model and average the predicted probabilities obtained from the different models. We then modified the MNISTNet class, where we set dropout\_at\_eval=True (this enables us to perform dropout also at validation, thus performing approximate Bayesian inference) and we changed the architecture of the network: as we did not obtain great results with the standard architecture, we decided to build a convolutional neural network with two convolutional layers and three fully connected layers, where dropout is performed in the fully connected layers. After trying different values of the parameter dropout\_p (the probability of setting the weight to 0 during dropout, we observed that 0.5 gave us the best performance). We then fine-tuned the regularization constant of the loss function and the best score was given by setting it to 0.0002, whereas we left batch\_size, num\_epochs and learning\_rate unvaried, obtaining an overall performance of 2.139.