Deep Ensemble Neural Networks

Praveen Kalva spkalva3

Group Partner: Anes Kim anesk2

PSYC 489 Section UG

Introduction

In machine learning, achieving reliable predications across varied datasets remains a challenge, but ensemble methods have emerged as a powerful solution. Ensemble techniques use multiple models, instead of a single model, and combine their predictions. Ensemble models help reduce variance in predictions and typically outperform any of the individual models. In deep learning, neural networks have very high variance because of the stochastic training process that can lead to vastly different weights per training run, and thus varying predictions. By combing the results of neural networks, ensembles can "reduce the variance of predictions and generalization error" (Brownlee 2019).

Among ensemble techniques, bagging is particularly notable for reducing variance and avoiding overfitting. Bagging trains multiple models on different subsets of the total training data, each randomly sampled with replacement, and then averages the models' predictions to make a final prediction. This method induces diversity into the training as each model gets trained by a different sample of the data, which leads to more reliable predictions. While each individual model might have some variance and error, the aggregation of their predictions reduces the total variance and error and increases generalization accuracy.

The performance of bagging is influenced by several factors including the number of models in the ensemble, the bagging size (size of the training subsets), and the individual model complexity. Understanding how these factors affect the performance of ensembles will help to optimize their application on different tasks. This study explores the effects of the number of models, bagging size, and model complexity (through hidden layer size) on neural network ensembles by evaluating test performance on the MNIST dataset.

Methods

Dataset and Preprocessing:

The dataset used in the experiment is the popular MNIST handwritten digits dataset. Each training sample is a 28x28 pixel image of a handwritten digit from 0 – 9. The goal is to classify images to their corresponding digit (0 – 9) with high accuracy. We split the dataset into the following training/validation/test set sizes:

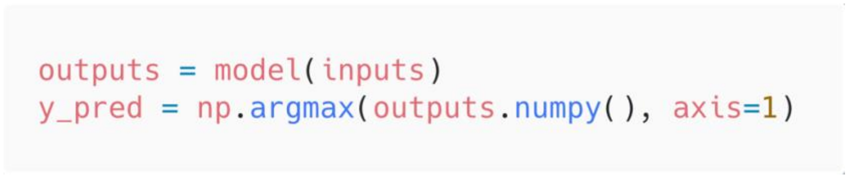
- Training Set: 50,000 samples
- Validation Set: 10,000 samples
- Test Set: 10,000 samples

All samples were flattened from a 28x28 matrix to a 784-dimensional vector for easy input into our neural networks. Additionally, pixel values were normalized to lie between the range of 0 – 1.

Individual Learners (Simple Neural Network):

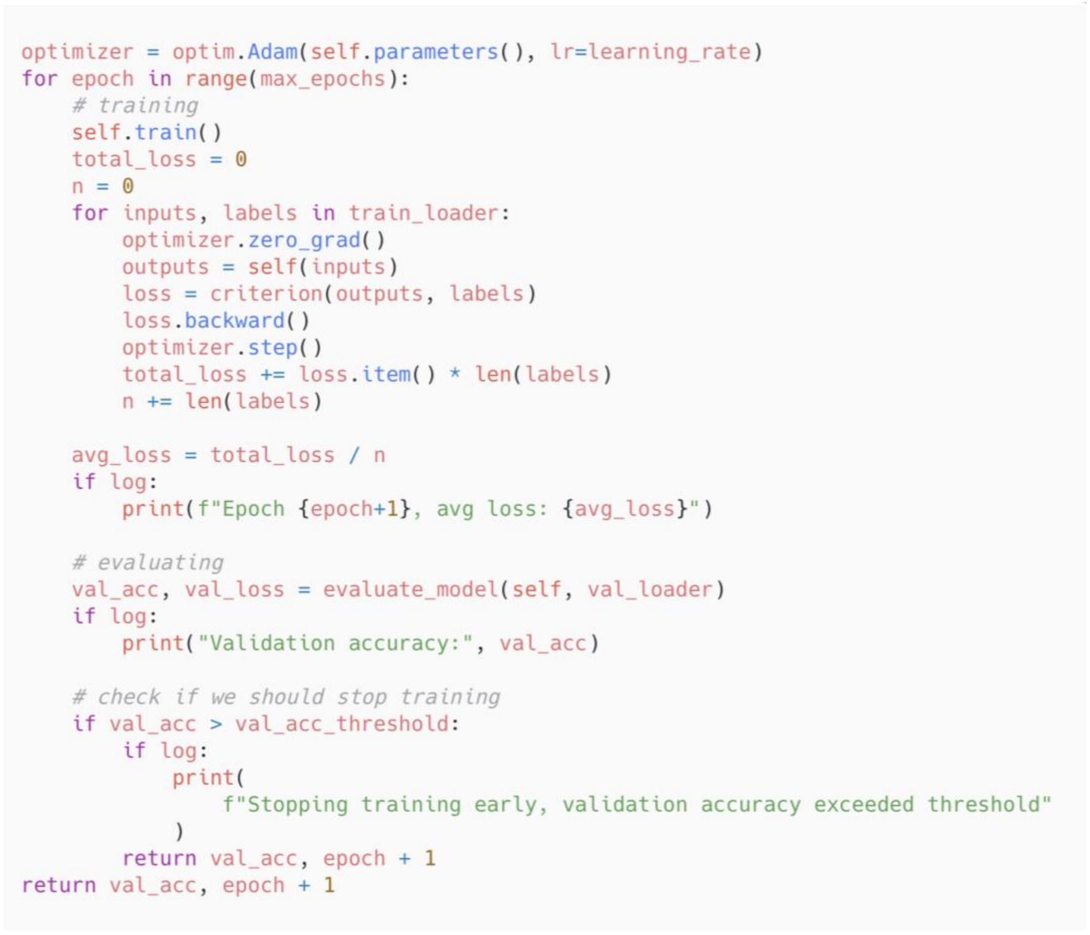
Within the ensemble network, there are multiple individual learners/models whose predictions are aggregated. The individual learners used within this experiment are simple 3-layer neural networks, consisting of an input layer, hidden layer, and output layer. The input layer has a dimension of 784 to fit the flattened MNIST images. The hidden layer size is a hyperparameter that has to be set before creating the network. Finally, the output layer has a dimension of 10 for the 10 different digit labels.

Each unit in the output layer represents the confidence that an input sample is a certain digit. When a test sample is fed through the network, the output layer represents the confidence that the image is each of the 10 digits. So, the final prediction/classification is the highest activated unit in the output layer. The following code describes this:



```
outputs = model(inputs)
y_pred = np.argmax(outputs.numpy(), axis=1)
```

Additionally, this network uses RELU for the activation function. It also uses Cross Entropy Loss (also known as Logarithmic Loss) as the loss function, which measures the difference between the discovered probability distribution of the model and the true distribution of the labels. Cross Entropy Loss is effective for classification tasks as it pushes the model to produce probability distributions that match the actual labels, leading to more accurate and reliable predictions. This network uses the ADAM learning optimizer with a starting learning rate of 0.01. The network also has an early stopping feature that can stop training early if it meets either of 2 conditions: trains for a maximum number of epochs (*max_epochs*) or passes a minimum accuracy threshold on the validation set (*val_acc_threshold)*. Early stopping reduces training time and is utilized by the ensemble to prevent long training on any one individual learner.



```
optimizer = optim.Adam(self.parameters(), lr=learning_rate)
for epoch in range(max_epochs):
    # training
    self.train()
    total_loss = 0
    n = 0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = self(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * len(labels)
        n += len(labels)

    avg_loss = total_loss / n
    if log:
        print(f"Epoch {epoch+1}, avg loss: {avg_loss}")

    # evaluating
    val_acc, val_loss = evaluate_model(self, val_loader)
    if log:
        print("Validation accuracy:", val_acc)

    # check if we should stop training
    if val_acc > val_acc_threshold:
        if log:
            print(
                f"Stopping training early, validation accuracy exceeded threshold"
            )
        return val_acc, epoch + 1
return val_acc, epoch + 1
```

Ensemble Model (Aggregate the Individual Learners):

The ensemble model utilizes multiple individual learners, who are basic neural networks, and their individual predictions are averaged and aggregated together to make a final classification. This ensemble class requires a parameter: *num_models*, which is the number of individual learners to be used.

When making predictions with the ensemble model, each individual learner or basic neural network will get passed the input. The output vector (dimension=10) of each network is then aggregated by taking the mean of each feature. The result is a mean output vector whose dimension is still 10, describing the average confidence among the individual learners that the input is each of the 10 digits. From here, a similar process is used to make the final classification:

the final classification is the highest activated unit in the mean output vector. The following code shows the aggregation of results from each of the individual learners:

```
def forward(self, x):
    outputs = [model(x) for model in self.models]
    return torch.stack(outputs).mean(dim=0)
```

Training is done iteratively, where each individual learner will get trained independently. Bagging or bootstrapping is used to vary the data that each learner will get trained with. In particular, each basic neural network will get trained with a subset of the total training data. The size of the subset is a training parameter: *bagging_size*, and the subset is generated by randomly sampling with replacement from the training dataset. Bagging promotes diversity in training which can help prevent overfitting, and bagging across many models in an ensemble makes predictions more robust to noise/outliers. Increasing the *bagging_size* will increase the amount of training data each learner gets but also lengthen the training time. The following code shows the iterative training process and how bagging/bootstrapping is used:

```
for i, model in enumerate(self.models):
  # sample a subset of the training data (with replacement)
  idxs = np.random.choice(len(x_train), bagging_size, replace=True)
  x_train_subset = x_train[idxs]
  y_train_subset = y_train[idxs]
  train_loader = DataLoader(
      TensorDataset(
          Tensor(x_train_subset), Tensor(np.eye(10)[y_train_subset])
      ),
      batch_size=batch_size,
      shuffle=True,
  )
  # train the individual learner
  val_acc, epoch = model.fit(
      train_loader,
      val_loader,
      criterion,
      learning_rate,
      max_epochs,
      val_acc_threshold,
      log=False,
  )
```

Testing:

Testing was done to see how the number of models in the ensemble, bagging size, and hidden layer size affected performance of the model. We tested configurations of the following training parameters:

- Number of Models in the Ensemble (*num_models*): [1, 3, 5, 10, 20]
- Bagging Subset Size (*bagging_size)*: [500, 1000, 5000, 15000, 25000]
- Hidden Layer Size of Individual Learners: [64, 128, 256]

We kept the training data (50,000 samples), validation data (10,000 samples), and testing data (10,000 samples) the same throughout the experiment. Additionally, all models were trained with a starting learning rate of 0.01, a batch size of 256, and used the following early stopping training parameters:

- *Max_Epochs:* 50 epochs
- *Val_Acc_Threshold*: 0.95

Across all runs of the experiment, the test accuracy and test loss were reported.

Results

The study investigated the effect of varying bagging sizes and the number of models in the ensemble on the performance, measured by test accuracy and test loss. We also examined how different hidden layer sizes influenced these metrics. Figures 1 – 2 show how bagging size and number of models affect test accuracy and test loss respectively, in 64-unit hidden layer neural networks. Figures 3 – 4 are with 128-unit hidden layer neural networks and Figures 5 – 6 are with 256-unit hidden layer neural networks.

The figures reveal a trend in the data: as the bagging size increased, both the test accuracy and the reduction in test loss improved. This improvement was most significant with initial increases in the bagging size and showed diminishing returns when the bagging size was already substantial. Similarly, increasing the number of models in the ensemble consistently enhanced test accuracy. Notably, the jump from one model to three models resulted in large improvement. Beyond five models, performance still increased but at a lower rate, indicating diminishing returns. The experiments also revealed that increasing the hidden layer size had a positive impact on test accuracy and test loss, particularly at lower bagging sizes. One potential reason for this would be that the more complex model architecture helped extract and learn patterns from smaller training data. As the bagging size increased, the effect of larger hidden layer sizes became less noticeable, likely due to the models with smaller hidden layers having access to more training data.

The findings support the benefits of bagging and ensemble models in improving test performance. When compared to a single neural network (*num_models=1)*, our results consistently show that ensemble networks perform better with greater test accuracy and lower test loss. Additionally, increasing the bagging size and/or the number of learners in the ensemble can also help to improve model performance. However, there are diminishing returns to these parameters and increasing the bagging size or number of learners also greatly increases model complexity and training time, so these factors must be balanced.

Discussion

The findings from this experiment validate the theoretical benefits of bagging in ensemble methods, demonstrating that increasing the bagging size and the number of models can substantially improve performance, especially when compared to a single model. The study also identifies potential areas of further research or consideration to better understand the nuanced behavior of ensembles.

One of the key insights from this study is the diminishing returns when increasing the bagging size or number of models after a certain point. Although continuing to increase the bagging size and number of models theoretically improves the performance, in the real world, there are computational constraints. Further research could help identify a balance between computational resources and performance improvements. There could also be changes to the ensemble model to optimize the training time. In particular, since each model in the ensemble acts independently, training could be parallelized among the models. Similarly, predictions could be aggregated parallelly, potentially improving runtime significantly.

There are other ensemble techniques beyond bagging, such as boosting or stacking that offer alternative algorithms to reduce variance and error. Studies involving these methods could be compared to bagging to discover the benefits and costs of each ensemble technique, perhaps leading to a hybrid implementation that combines the best aspects of each method.

While this study provides effective results on the performance of bagging ensembles, it also opens several avenues for further research. By delving deeper into ensembles, we can learn more about and develop low-variance and high-performing machine learning models.

References

Rocca, J. (2021, March 21). Ensemble methods: Bagging, boosting and stacking. Medium. https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Brownlee, J. (2019, August 6). *Ensemble learning methods for deep learning neural networks*. MachineLearningMastery.com. https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/

https://github.com/pravshot/deep-ensemble-networks

Figures

Figure 1: Bagging Size vs Test Accuracy for Hidden Layer Size of 64

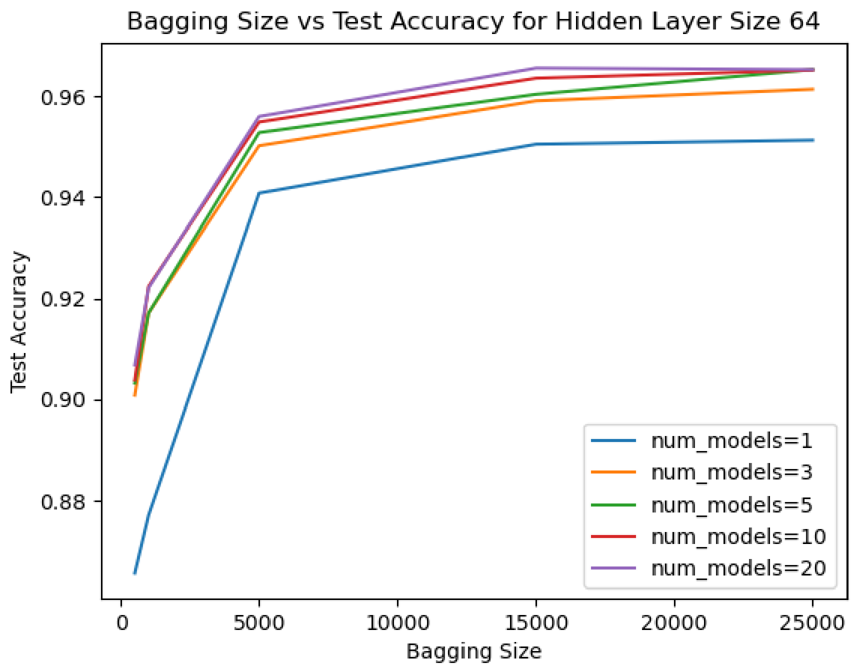


Figure 2: Bagging Size vs Test Loss for Hidden Layer Size of 64

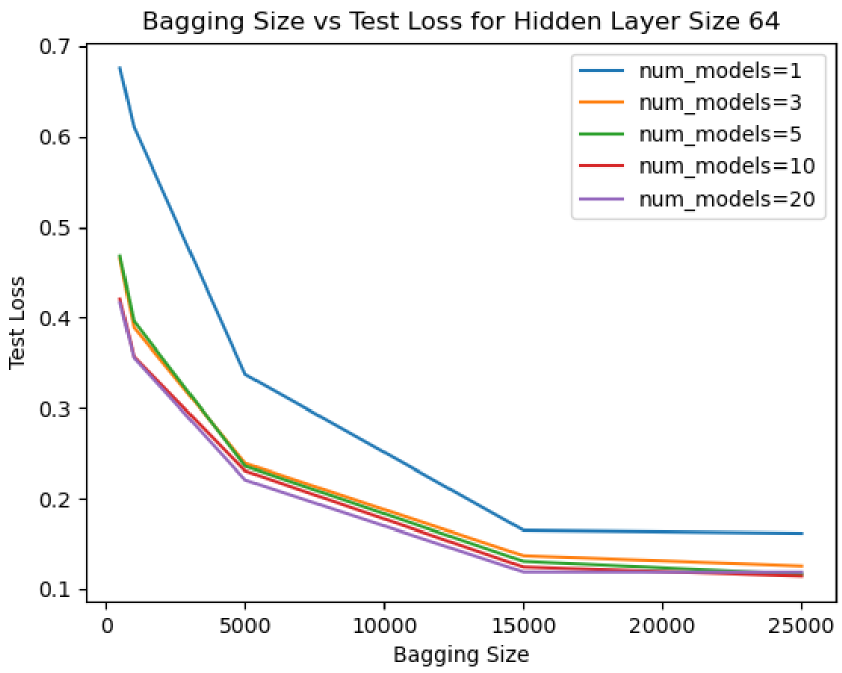


Figure 3: Bagging Size vs Test Accuracy for Hidden Layer Size of 128

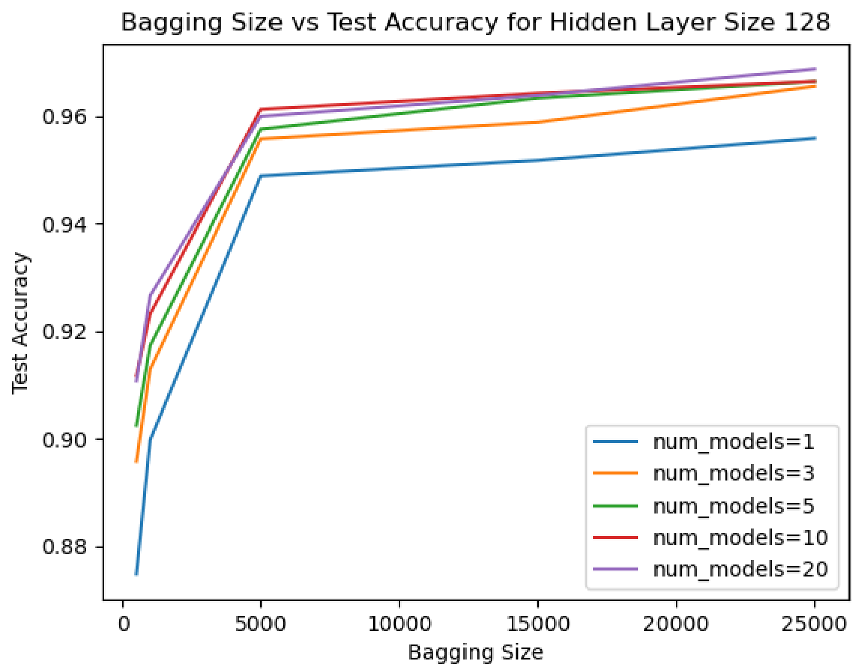


Figure 4: Bagging Size vs Test Loss for Hidden Layer Size of 128

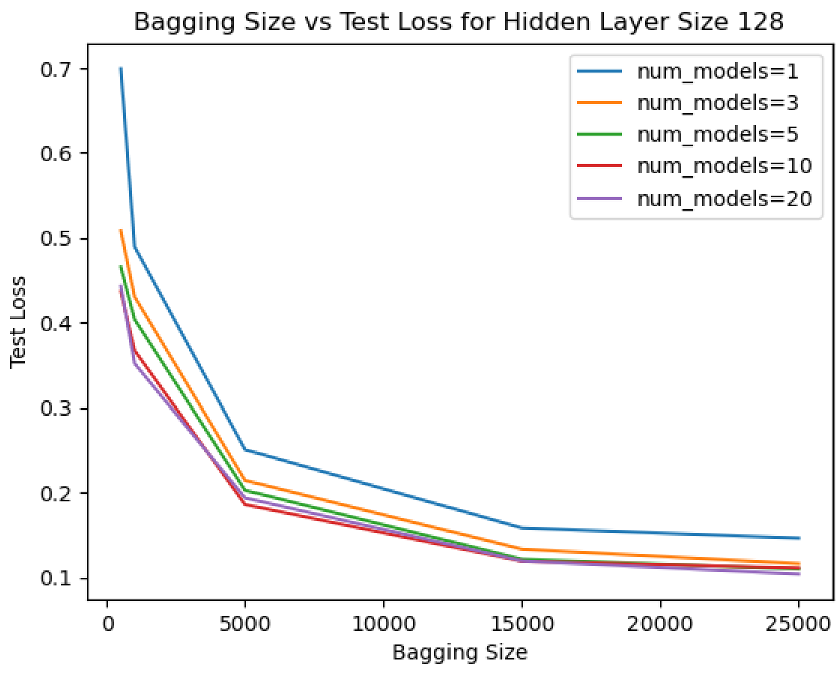


Figure 5: Bagging Size vs Test Accuracy for Hidden Layer Size of 256

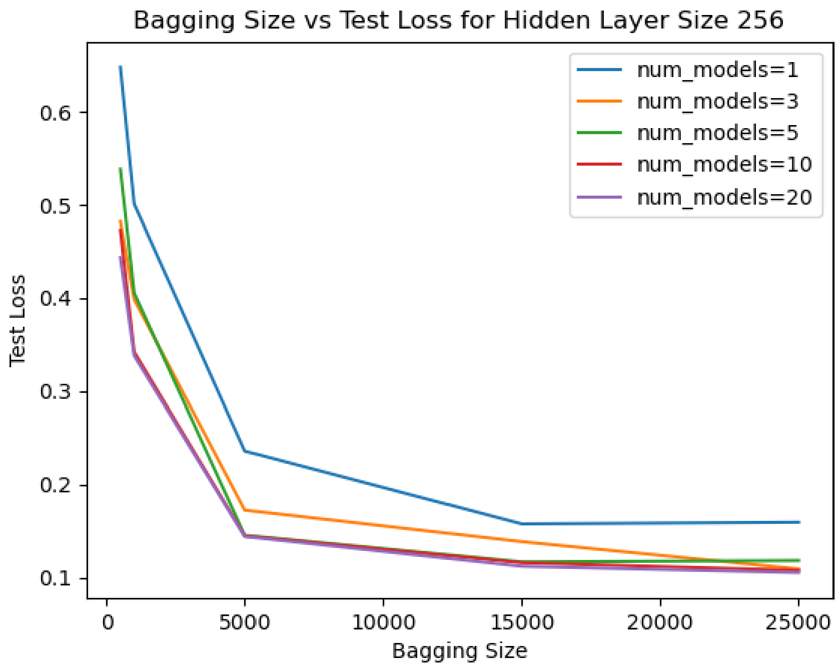
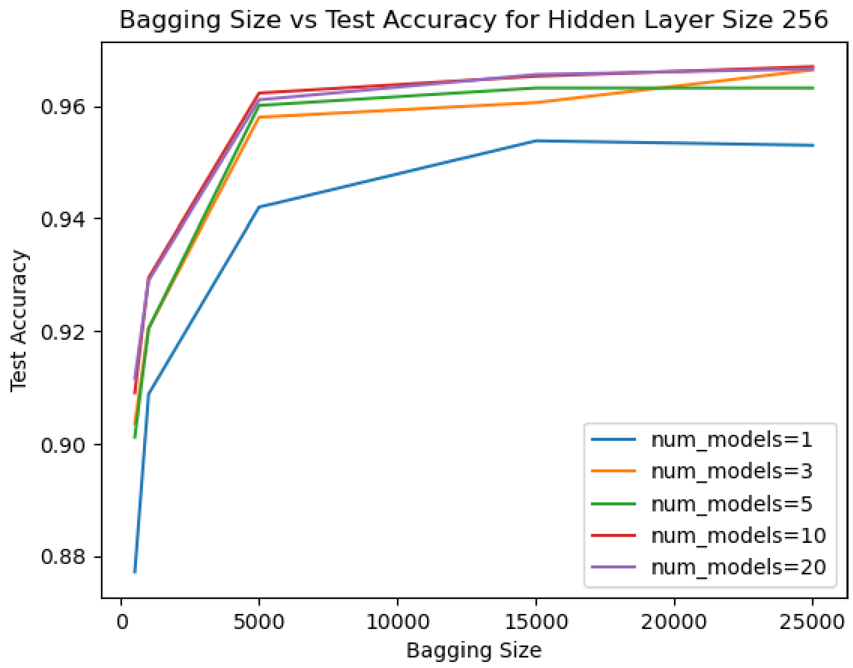


Figure 6: Bagging Size vs Test Accuracy for Hidden Layer Size of 256


Bagging Size vs Test Accuracy for Hidden Layer Size 256
Test Accuracy
Bagging Size
num_models=1
num_models=3
num_models=5
num_models=10
num_models=20