
BergNet: Classifying Harvard Dining Hall Food Using Deep Learning

Linda Zeng

Harvard University

Pre-College Program

Presented on August 2, 2024 | Revised on September 29, 2024

Contact: 26lindaz@students.harker.org

Keywords: Food Classification, Neural Networks, Machine Learning

Motivation

Annenberg Hall, known as “The Hogwarts of Harvard,” is Harvard’s iconic first-year dining facility, spanning 88,000 square feet. During my four weeks in the Pre-College Program, it became a central part of my daily routine. With its unique architecture and vibrant atmosphere, Annenberg plays a significant role in shaping the undergraduate and summer student experience.

In this project, I explore the potential of training a neural network to analyze and recognize the food served at Annenberg Hall. Leveraging machine learning techniques, I aim to assess how well a model can classify images of the diverse meals provided in the dining hall. This study not only serves as a fun and practical application but also introduces broader implications for machine learning in food analysis—ranging from nutrition recommendation and meal balancing to improving food monitoring in industries. This notebook serves as a guide through my experiments.

Data Collection

First, I collected images by visiting the dining hall on Wednesday, July 31, 2024, during lunch and dinner and capturing a total of 67 photographs of various food items. Each image was labeled into one of the following categories: “Vegetable,” “Grains,” “Soup,” “Fruit,” and “Protein.” This labeling process is crucial, as it allows me to create a structured dataset that the neural network can learn from.

I split the data using a 0.7 ratio, with 45 training data points and 22 test data points, while taking into account balancing classes. Given that this project is novel, I faced the challenge of working with a limited dataset that I had to collect myself. Data scarcity is a significant concern in the field of machine learning, as having a restricted number of examples can hinder a model's ability to generalize and accurately classify unseen data. To address this issue, I will employ data augmentation techniques and explore additional strategies to enhance my dataset, ensuring that my neural network can effectively learn to recognize the diverse array of food served at Annenberg Hall.

Data Augmentation

To increase the amount of training data samples, I provide a series of transformations to each training data sample in the training set. These included flipping them both horizontally and vertically, cropping them to a size of 100 by 100 pixels, and rotating them at angles of 45, 135, 225, and 315 degrees, translating the images slightly to the left and right, adding various types of noise (including random, Gaussian, and Poisson noise), boosting their colors for greater vibrancy, and adjusting the color balance to enhance brightness while creating darker and lighter versions of each image. The final augmented dataset was 765 data points. An example of results from data augmentation is shown in Figure 1.



Figure 1: Noisy, rotated, cropped, and original versions of a picture of corn, which is labeled as ‘Vegetable.’

Training a Custom Model with Data Augmentation Set

In this section, I focus on training our custom neural network model, (Annen)**BergNet**, using Mathematica’s NetChain functionality, which is displayed in Figure 2. BergNet consists of multiple layers, including two convolutional layers followed by ReLU activations and pooling layers, which help extract hierarchical features from input images. To prevent overfitting, I incorporate dropout layers with a value of 0.1. The network architecture transitions to fully connected layers to transfer the information learned from image features into the five classes. Through the softmax layer, each class receives a probability from the model, with the highest probability class being the model’s classification.

The model is trained on our augmented training set, which includes both the original images and their transformed versions. By presenting the model with these varied inputs during training, it learns to recognize patterns and features that are invariant to these alterations, thereby improving its ability to generalize to unseen data. Once the training is complete, the model’s performance is evaluated on a held-out test set that was not used during the training phase. This test set serves as an unbiased measure of how well the model can generalize its learned knowledge to new, unseen examples. By comparing the model’s predictions on the test set with the actual labels, I assess its accuracy.

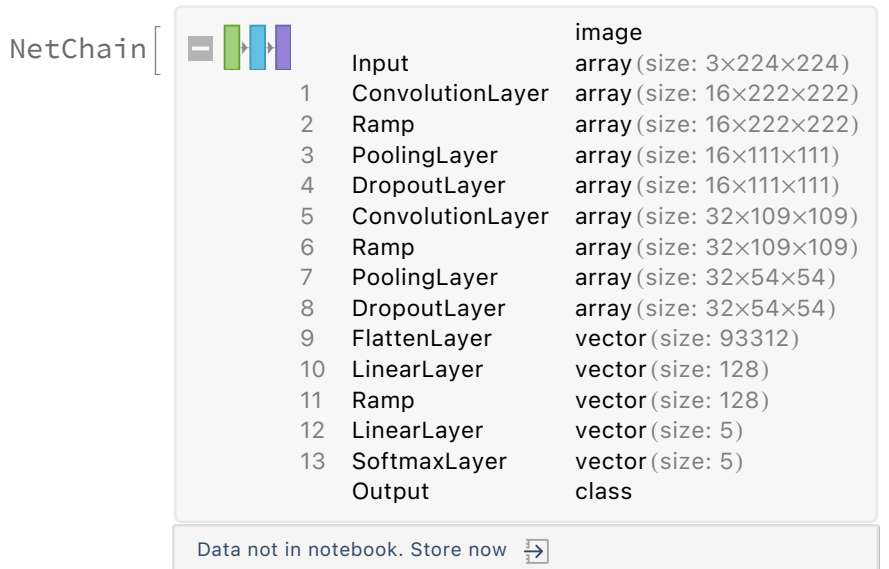


Figure 2. Architecture of custom BergNet, including corresponding sizes at each layer.

Investigating Performance without Data Augmentation


I also evaluate the impact of training our model without applying data augmentation techniques. Data augmentation typically helps improve the model’s generalization by introducing variability in the

training set, simulating different conditions the model might encounter. To investigate the importance of augmentation, I train a version of our model, BergNet, solely on the original, not augmented training dataset. All architecture and procedure remains constant, but the model is trained on only the 45 original samples.

Fine-tuning a Pre-trained Model

Given the limited size of our dataset, fine-tuning a pre-trained model is often a more effective approach than training a custom model from scratch. Pre-trained models, such as EfficientNet, have already been trained on vast datasets like ImageNet, allowing them to learn general features that are useful for a wide variety of tasks. By fine-tuning, we leverage the knowledge these models have acquired and adapt it to our specific problem with a smaller amount of data. This method helps mitigate overfitting, which is a common risk when training models on limited data, as the pre-trained model starts with well-established weights that only need to be slightly adjusted. This results in faster convergence and often leads to better performance compared to training a custom model that starts from random initialization.

I load in pre-trained EfficientNet, take the weights, parameters, and layers for most of EfficientNet but adjust the final output linear layer and softmax to output five classes for our specific task. I also replace the input and output without custom encoder and decoders.

NetChain []

| | | |
|---|---------------------|-------------------------|
| | Input | image |
| | | array (size: 3×224×224) |
| 1 | NetChain (24 nodes) | vector (size: 1280) |
| 2 | LinearLayer | vector (size: 5) |
| 3 | SoftmaxLayer | vector (size: 5) |
| | Output | class |

| | | |
|--------------------|-------------------------|--------------------------|
| 1: NetChain | | |
| | Input | array (size: 3×224×224) |
| stem_conv | ConvolutionLayer | array (size: 32×112×112) |
| stem_bn | BatchNormalizationLayer | array (size: 32×112×112) |
| stem_activation | LogisticSigmoid[x] x | array (size: 32×112×112) |
| block1a | NetChain (6 nodes) | array (size: 16×112×112) |
| block2a | NetChain (9 nodes) | array (size: 24×56×56) |
| block2b | NetGraph (11 nodes) | array (size: 24×56×56) |
| block3a | NetChain (9 nodes) | array (size: 40×28×28) |
| block3b | NetGraph (11 nodes) | array (size: 40×28×28) |
| block4a | NetChain (9 nodes) | array (size: 80×14×14) |
| block4b | NetGraph (11 nodes) | array (size: 80×14×14) |
| block4c | NetGraph (11 nodes) | array (size: 80×14×14) |
| block5a | NetChain (9 nodes) | array (size: 112×14×14) |
| block5b | NetGraph (11 nodes) | array (size: 112×14×14) |
| block5c | NetGraph (11 nodes) | array (size: 112×14×14) |
| block6a | NetChain (9 nodes) | array (size: 192×7×7) |
| block6b | NetGraph (11 nodes) | array (size: 192×7×7) |
| block6c | NetGraph (11 nodes) | array (size: 192×7×7) |
| block6d | NetGraph (11 nodes) | array (size: 192×7×7) |
| block7a | NetChain (9 nodes) | array (size: 320×7×7) |
| top_conv | ConvolutionLayer | array (size: 1280×7×7) |
| top_bn | BatchNormalizationLayer | array (size: 1280×7×7) |
| top_activation | LogisticSigmoid[x] x | array (size: 1280×7×7) |
| avg_pool | AggregationLayer | vector (size: 1280) |
| top_dropout | DropoutLayer | vector (size: 1280) |
| | Output | vector (size: 1280) |

Data not in notebook. Store now 

Figure 3. Architecture of fine-tuned Efficient Net on Mathematica, which has a hierarchical structure denoted by more NetChains and NetGraphs, each including layers. We focus on using the first 24 nodes of EfficientNet's NetChain and adding a LinearLayer and SoftMax Layer to output five classes.

Results

Testing on Data from a Different Day

I returned to the dining hall on Thursday to capture another set of food images, collecting 20 new photos that will serve as an alternative test set. This additional test set allows us to assess how well our models generalize when exposed to food from a different day, which may have slight variations in appearance, lighting, or presentation. By evaluating the models on this new set, we can gauge their robustness and ability to handle subtle differences in the data, which is crucial for real-world applica-

tions where conditions are rarely identical to those in the original training environment.

Summary of Results

Shown in Table 1, our results show that training BergNet on data improves significantly from no training even if data is limited. In particular, however, data augmentation further increases results from 45.5% to 63.6%. Considering that BergNet is trained on only 765 data points, which are sourced from only 45 original data points, its 63.6% accuracy is notable. We find that BergNet trained on the augmented dataset also showed improved generalizability to the Thursday test set.

Fine-tuning EfficientNet showed to be the most effective on this task, with results rising to 72.7%. This shows how the complex architecture and past pretrained knowledge allowed EfficientNet to better perform classification in this low-resource task. Results are primarily attributed to training, as EfficientNet’s zero shot score is still low. Furthermore, EfficientNet showed high generalizability to Thursday data compared to BergNet, decreasing by only 3% in accuracy. However, since we are limited by data scarcity, there is a margin of error that must be considered as test sets are relatively small.

Out[499]=

| Model | Wednesday Accuracy (%) | Thursday Accuracy (%) |
|-----------------------------------|------------------------|-----------------------|
| BergNet without Training | 18.2 | 25. |
| BergNet | 63.6 | 40. |
| BergNet without Data Augmentation | 45.5 | 25. |
| EfficientNet without Fine-tuning | 13.6 | 35. |
| Fine-tuned EfficientNet | 72.7 | 70. |

Table 1. Accuracy on test data from initial dataset (Wednesday) and from an additional test set (Thursday).

Analysis

Predictions of Fine-tuned EfficientNet on Thursday Data

Our best-performing model, EfficientNet, also showed very high capability to generalize to new data that does not directly correlate with its training data. We found that it primarily correctly classified “everyday foods,” foods which were found on both Wednesday and Thursday in the dining hall, as seen in Table 2. This makes sense since it had already been trained on similar pictures. Examples of these foods are apples, rice, noodles, and cookies, which it predicted correctly. If we search through the original data, we can find similar images as the ones shown below in the Thursday test data.

Out[]=



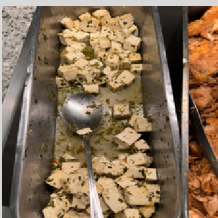
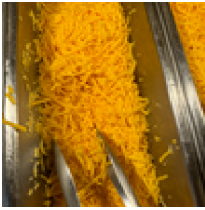
| Image | Predicted Label | Actual Label |
|---|-----------------|--------------|
|  | Soup | Grains |
|  | Grains | Protein |
|  | Fruit | Fruit |
|  | Fruit | Fruit |
|  | Protein | Protein |

Table 2. Example predictions by fine-tuned EfficientNet on Thursday test data.

Case Studies

While the first image shown in Figure 4 is of cheese, which is labeled as soup. The second image is of carrots, which is labeled as vegetables. However, the task is evidently difficult since there is little to

distinguish the two.



Cheese, Labeled as a Vegetable

{"Fruit" → 0.00119143, "Soup" → 0.0524874, "Vegetable" → 0.945291}



Carrots, Labeled as a Vegetable

{"Fruit" → 0.00119143, "Soup" → 0.0524874, "Vegetable" → 0.945291}

Figure 4. Images of cheese and carrots and their probabilities for each class generated by fine-tuned EfficientNet.

While these two images in Figure 5 are both labeled soup, the pretrained model got the redder one incorrect as vegetable and the second one correct as soup. This may be because of the brighter color of the first one. This shows how the system has discrepancies and biases likely due to overfitting on limited training data.



Soup, Labeled as a Vegetable



Soup, Labeled as a Soup

Figure 5. Images of soup and their labels predicted by fine-tuned EfficientNet.

Conclusion

In conclusion, this project demonstrates the effectiveness of neural networks in classifying food items within a unique and limited dataset. While my fine-tuned EfficientNet model outperformed BergNet and achieved 73%, it is noteworthy that BergNet achieved significant performance levels despite being trained on only 765 images. The incorporation of data augmentation techniques proved instrumental

in enhancing model performance, resulting in an increase from 45% to 63%. Additionally, my fine-tuned EfficientNet model generalized exceptionally well to images from a different day, achieving a similarly high accuracy of 70%. Overall, I found data augmentation and fine-tuning serve as effective methods in working with low amounts of data. Furthermore, the more consistent the food served at Annenberg, the better BergNet performs. These findings not only provide a roadmap for developing a custom Mathematica-based neural network tailored to a low-resource dataset but also hold the potential to enhance the dining experience for Harvard students at Annenberg Hall.