The Application and related Experiments for CNN model on Food Image Classification

Wenbo Hu, Wenxiao Li, Huaning Liu
University of California, San Diego
`w1hu@ucsd.edu, wel032@ucsd.edu, h9liu@ucsd.edu`

Abstract

In this project, based on the given food-101 image data, we build a CNN model, vgg-16 model, and resnet 18 model to perform image classification on the data. Also, related experiments on different parameters as well as architectures are attempted on CNN model to optimize its performance of classification on the test set. Furthermore, more visualizations regarding features maps, weight maps, and the models' performances on the split datasets during the training process are present for discussion and exploration. In this case, our baseline model, built by writeup requirements, reports a test accuracy of about 38.8%; after further hyper tuning and architecture optimizing, a customized model improves the test accuracy to be 49.2%. And for the other two models, vgg 16, after hyper tuning, brings a test accuracy of about 70.9%; while the resnet 18 shows a test accuracy of about 68.1%. Noticeably, comparing with the baseline model, there exists obvious improvements in the test accuracies for other models, which is reasonable and expected. Further analysis and explorations will also be presented regarding the cases here.

1 Introduction

CNN model is widely used in the Computer Vision area when it comes to image classification; this time, given the Food 101 dataset, we would like to implement and explore how good CNN model, as well as vgg 16 and resnet 18 model, could perform on image classification of this dataset. Also, proper evaluation metrics (accuracy in this case) are picked to represent the related performance. All model frameworks here are formulated with PyTorch, which is a very efficient and useful deep learning library in python. By examining and comparing the classification accuracy for different models (after tuning), we can hopefully draw some analytics and conclusions from our experiments as well as the training process.

1.1 Dataset Introduction

This is a dataset consisting of 101,000 images collected and developed by a research team of ETHZ. The images are collected from foodspotting.com, which is a site that allows its users to share what they're actually eating by shooting images, from which the top 101 popular kinds of dishes are selected and pushed into this dataset. Some famous dishes, including steak, cupcake and ice cream, etc, are shooted from different angles as well as under different levels of lights, therefore leading to various images. For each dish, 750 training images and 250 test images are samples, which means there will be equal 1000 images for each dish. Figure 1 gives a glimpse into these dishes to see what they actually look like.

Figure 1: Glimpse on sampled image for each dish

2 Related Works

Besides the specific topic we are discussing in this project, there are other works that also intervene with the application of neural network (CNN) models on the food 101 dataset. In Joshua Ball's paper, **Food Classification with Convolutional Neural Networks and Multi-Class Linear Discernment Analysis** [1], a robust CNN model as well as a Linear Discernment Analysis (LDA) is implemented and whose performances on the test set are accordingly compared and analyzed. Compared with our work, an additional PCA is assigned at the beginning to reduce the dimension and optimize the training process. Furthermore, other cases such as binary classification are subsetted to discuss and compare with the full classification process to thoroughly discuss the merit and limitations of CNN and LDA. Comparatively, our work doesn't take too much on the adjustment of the dataset, but we generate more models to compare their performances accordingly (vgg 16, resnet 18).
On the other hand, Niki, Gian, and Christian's work, **Wide-Slice Residual Networks for Food Recognition** [2], constructed another learning network (residual learning) to examine its performance on the test set. Also, some parameters, as well as inner methodologies, are discussed to compare with the training results. Compared with our fine-tuned CNN model, the overall structure of their wide-slice residual network is deeper, which would possibly take a longer time to train given the same device. Furthermore, given that we directly use accuracy as evaluation metrics in our project, Niki's team reports both Top-1 recognition accuracy and Top-5 criterion, which would comparatively evaluate the models in a more fair perspective. Overall speaking, this paper focuses more on the design of their network algorithm as well as the potential benefits of their proposed solutions.

3 Models

3.1 Baseline Model

For the baseline model, it is generated with required parameters as well as architecture on the writeup, such that four convolutional layers (with BN and ReLU) as well as two pools, including maximum pool and average pool, and another two fully connected layers are included to be the overall structure. After the related adjustments of data dimensions, the networks run and report a baseline test accuracy of about 38.8%. During the training process (loss plots and accuracy plots shown on Figure 2), a smooth decrement and increment for train loss and train accuracy are present; while for the validation loss and accuracy, the curve trembles a little bit at the very end. For now, the training accuracy and validation accuracy, accordingly, is reported to be 35.8% and 36.3%, which all accuracies (including training, validation, and esp. test) will be regarded as the benchmark in the following optimization, fine-tuning, and different model training processes.

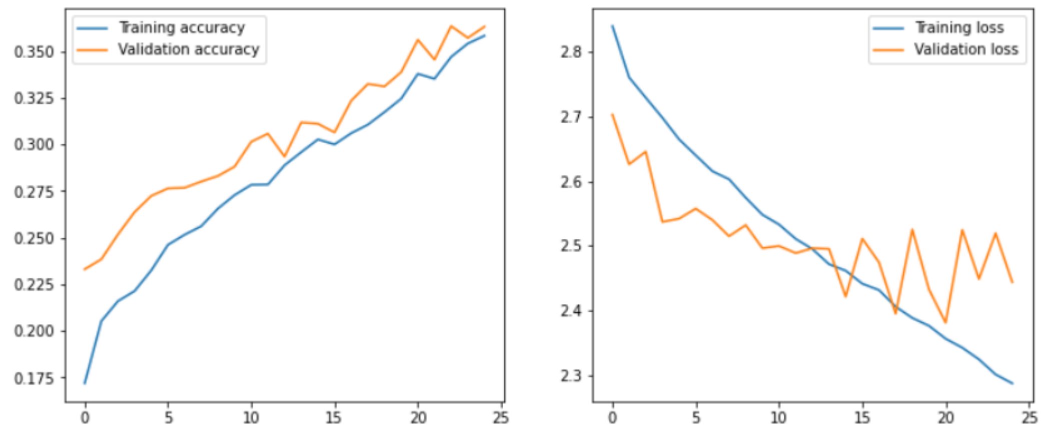

Figure 2: Accuracy Plot and Loss Plot for baseline model

3.2 Custom Model

A quick guess for a not-so-good result of the baseline model might be: less number of epochs, model being not complicated enough to distinguish between different categories, or bad parameters. For the custom model, after many attempts, we settle the development of architecture to be adding another two convolutional layers after conv3 and another dropout layer after conv3; meanwhile, from the perspective of parameters, we change the learning rate to be half of the baseline model while increasing the number of epochs to be 40 to make it converging. Additionally, batch size is adjusted to be 8. Under this circumstance, the test accuracy is reported to be 49.2%, with a train accuracy of 44.8% and final validation accuracy of 46.5%, which infers a big increment compared with the baseline model. Related architecture table is presented on Figure 3. Related loss plot and accuracy plot is presented on Figure 4.

Particular Layer	Layer dimension (in-out)	kernel size	stride
conv	3 - 64	3	1
conv	64-128	3	1
conv	128-128	3	1
dropout	128-128	NA	NA
conv	128-128	3	1
conv	128-128	3	1
maxpool	128-128	3	1
conv	128-128	3	1
avgpool	128-128	3	1
nn.Linear	128-128	NA	NA
dropout	128-128	NA	NA
nn.Linear	128-20	NA	NA

Figure 3: Custom Model Architecture

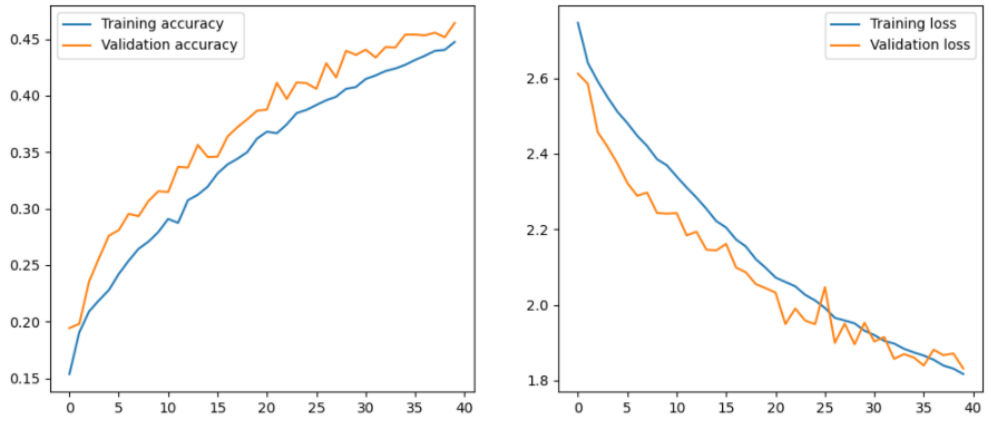

Figure 4: Accuracy Plot and Loss Plot for custom model

3.3 Resnet 18

For the Resnet 18 model, overall speaking, before fine-tuning and after fine-tuning, it performs better than the custom model we built on the previous part. Following the definition of Resnet 18, we built this model with the activation function to be ReLU to train on the dataset. In the beginning, the model without tuning gives a test accuracy of about 60.5%, with its training accuracy increasing to be about 66% and validation accuracy to be 61%, which infers a not powerful result here, with the validation accuracy also trembling a lot during the training process. After fine-tuning, with the slight increment of the learning rate from 0.001 to 0.002, increment of batch size from 8 to 24, and replacement of Adam to be SGD with a momentum of 0.9, the test accuracy increases to be 68.1%, with a more smooth train accuracy and validation accuracy for 70% and 66%. Related loss plot and accuracy plot is presented on Figure 5.

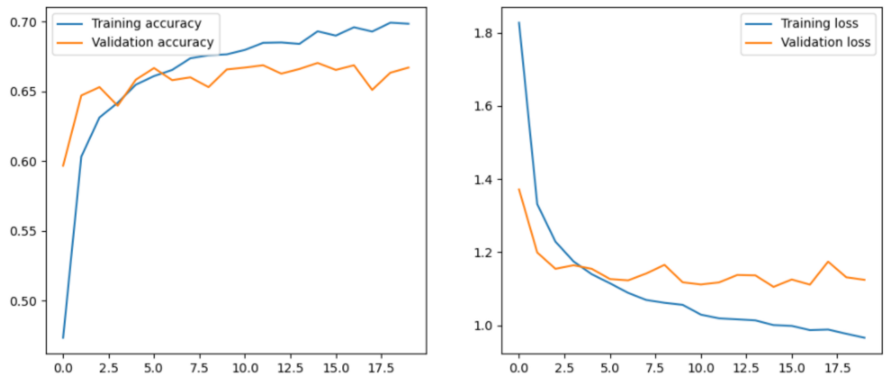

Figure 5: Accuracy Plot and Loss Plot for Resnet 18

3.4 VGG 16

For the VGG 16 model, we implement it by freezing all the parameters of the convolutional layers. For the classifier layer, we directly connect a fully-connected layer with 20 output units. This implementation is based on the observation that: the original torchvision model consists of several fully connected layers, with about 10^6 parameters in each layer. However, if we directly use the default architecture, the training time for each epoch will be too long, and the accuracies converge in epoch 2-3, thereby making it extremely difficult to observe any changing patterns. Related loss plot and accuracy plot is presented on Figure 6.

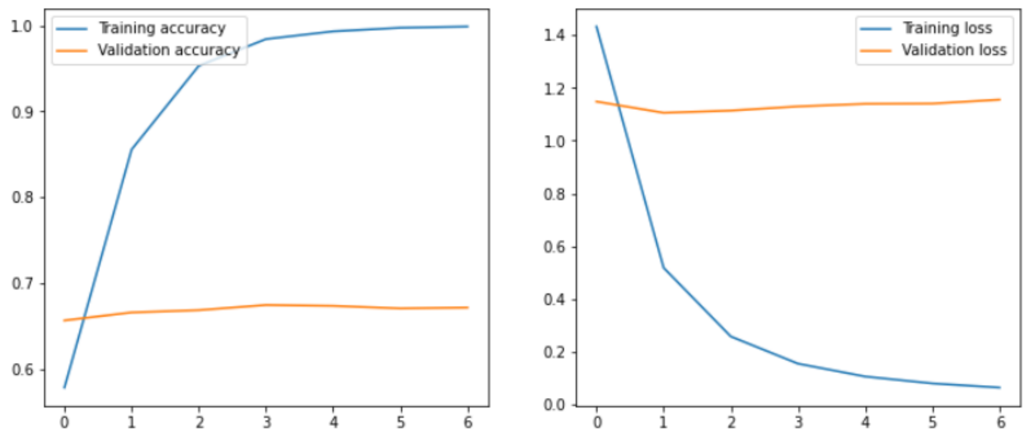

Figure 6: Accuracy Plot and Loss Plot for VGG 16

4 Experiments

4.1 Custom Model

For the baseline model, we are able to get test accuracy of about 38.8%, the training accuracy and validation accuracy, accordingly, are 35.8% and 36.3%. Here is the plot for the training procedure.

For the custom model, we implement mainly 3 types of changes: increase the number of layers, adjust hyperparameters, and add more epochs. Here is the table of how we combine these changes together. Below shows how test accuracy increases to 49.2% based on changes. Notice, the first row of the table is the baseline model. The last row of the table is the best custom model. Related experiments are summarized in 7

learning rate	epochs	RanEqualize?	Add 2 layers?	Test accuracy	Train Accuracy	Validation Accuracy
0.001	25	No	No	38.8%	35.8%	36.3%
0.001	25	No	Yes	42.6%	38.6%	40.1%
0.001	25	Yes	Yes	43.4%	39.2%	41.8%
0.0005	25	Yes	Yes	43.6%	38.7%	41.1%
0.0005	40	Yes	Yes	49.2%	44.8%	46.5%

Figure 7: Custom Model Experiment

4.2 Resnet 18

For the transfer learning with Resnet 18 model, the best model we have found through fine-tuning will be "lr = 0.002, epoch = 20, batch size = 24", and with SGD optimizer as data augmentation. In this case, we are able to achieve a training accuracy of 70% and validation accuracy of about 66%. The test accuracy is 68.1%. For the experiment process, I have tried different numbers of epochs to see which value benefits the most for converging and different learning rates to check the result. Also, choosing SGD or Adam is another important decision in the experiment, where we finally give an answer of SGD with a momentum value of 0.9, related experiments are summarized in 8.

learning rate	epochs	batch size	Optimizer	Test accuracy	Train Accuracy	Validation Accuracy
0.001	30	8	Adam	49.1%	95.4%	48.5%
0.001	10	8	Adam	40.2%	60.0%	55.4%
0.001	20	24	Adam	50.5%	58.0%	64.5%
0.002	20	24	SGD	68.1%	70%	66%
0.002	30	24	SGD	59.4%	55.8%	57.7%
0.0005	20	24	SGD	67.3%	65.8%	63.6%

Figure 8: Resnet 18 Model Experiment

4.3 VGG 16

For transfer learning with Vgg 16 model, the best model we have found through fine-tuning is "lr = 0.0005, epoch = 7, batch size = 4, and with SGD optimizer as data augmentation." In this scenario, we are able to reach 99.9% Train Accuracy, and 67.1%. Validation Accuracy by the end of epoch 7. The test accuracy for the model is 70.9%. Also, we have tried different sets of hyperparameters: when epoch = 25, the model will strongly overfit, with Train accuracy 100%, Validation accuracy 66.9%, and test accuracy 69.6% eventually. The validation accuracy makes no progress, from epoch 7 to 25. When we use Adam optimizer, we get 83.5% Train Accuracy, 62.8% Validation accuracy, and test accuracy 68.8%. When we use batch size = 24, instead of 4, we get 90.6% Train Accuracy, 67.4% Validation accuracy, and test accuracy 69.8%. Refer to the table for accuracies: 9.

learning rate	epochs	batch size	Optimizer	Test accuracy	Train Accuracy	Validation Accuracy
0.0005	7	4	SGD	70.9%	99.9%	67.1%
0.001	7	4	SGD	70.0%	96.8%	68.1%
0.0005	25	4	SGD	69.6%	100%	66.9%
0.0005	7	4	Adam	68.8%	83.5%	62.8%
0.0005	7	24	SGD	69.8%	90.6%	67.4%

Figure 9: VGG 16 Model Experiment

5 Feature Map and Weights Analysis

For weights maps of the first layer, our custom model and vgg 16 weight maps are smaller than resnet 18 weight map because of different model architecture. The visualization shows that weights are activated very differently. They aimed to summarize different features in the data. For feature maps. In their visualizations, all the feature maps are similar to and can still be recognized as the original image since they are still very representative of original features in the image. For all three networks, the first layer feature map, middle layer feature map and final layer feature map follows certain properties. The first layer feature map is the most clear and similar to original image since the only one convolution is processed. The middle layer feature map is considering different feature expressions of the image, such as sharpen, blurry, shape detector and so on. And the final feature map is the most blurry compared to the original since they are the most global generalized features from the image.

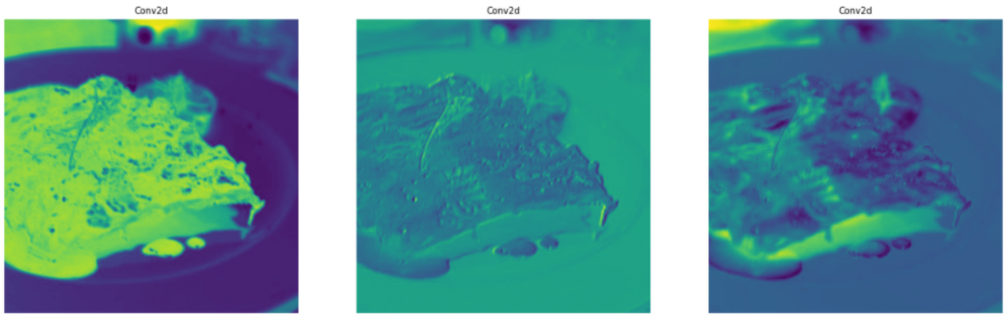

Figure 10: Feature Map for Custom Model

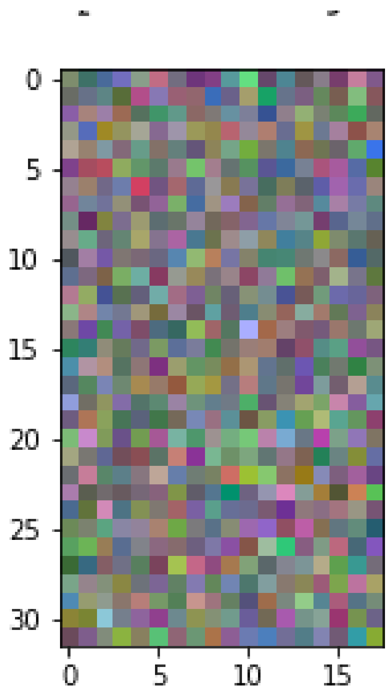

Figure 11: Weight Map for Custom Model

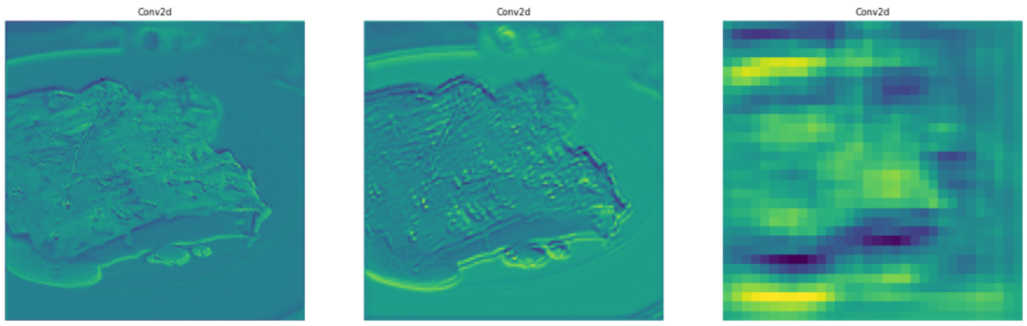

Figure 12: Feature Map for Resnet 18 Model

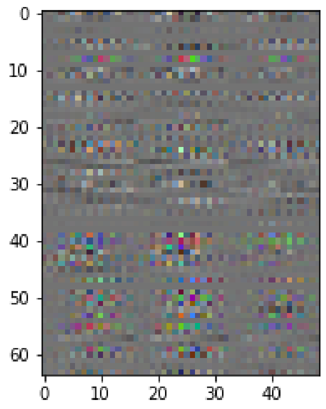

Figure 13: Weight Map for Resnet 18 Model

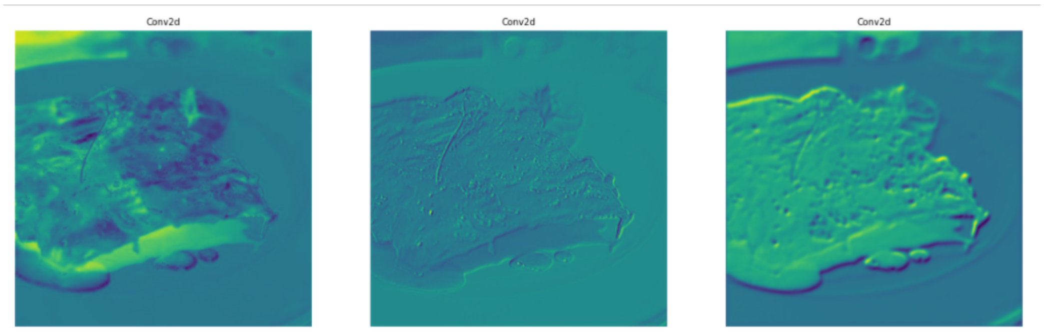

Figure 14: Feature Map for VGG 16 Model

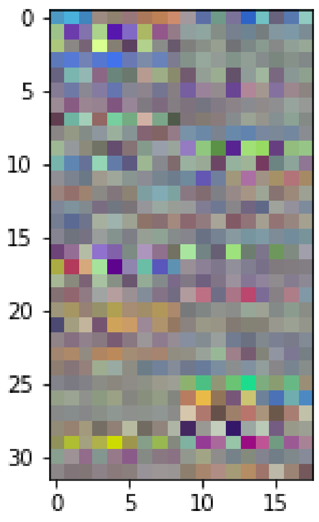

Figure 15: Weight Map for VGG 16 Model

6 Discussion and Findings

Notice that when we train the baseline model, the validation accuracy is higher in this case. This is due to the existence of a dropout layer at the end. When training, some weights will be dropped, giving more difficulty to obtain a correct output for a training data. However, in the validation process, the dropout layer is disabled, so in validation, we can utilize all the weights to produce predictions.

Based on test accuracies, baseline has 38.8%, custom model has 49.2%, Resnet 18 has 68.1%, Vgg 16 has 70.9%. Based on the architecture and parameters of Vgg 16 and Resnet 18, we can find that if we can increase the number of convolution layers, we will have better testing accuracies. Adjusting the model to a suitable number of epochs, with a well-designed learning rate will also give rise to better results of the custom model.

For the custom model, we implement mainly 3 types of changes: increase the number of convolutional layers, adjust hyperparameters, and add more epochs. Conclusion. Increasing the number of layers will make the model more complicated, thus making it more capable of distinguishing some principal components from different categories. Adjusting the hyperparameters (e.g. learning rate) will make the model make better adjustments when doing gradient descent. In our case, we decrease the learning rate when the model has deeper layers. Adding more epochs will give the model more iterations to explore, in order to update its weights to make more accurate predictions.

In transfer learning, we need to pay more attention to the type of optimizers that we use. In the training procedure of Resnet 18, using SGD optimizer will give significantly better results than using Adam optimizer. We also need to control the number of epochs. Since both Resnet 18 and Vgg 16 are deep networks, running more epochs may cause overfitting problems. A good choice of number of epochs can cope with this issue. Also, as stated in the experiment sections, we modified the number of fully connected layers to reduce the number of parameters when using Vgg 16, to slow the convergence speed in order to better monitor the changing of losses and accuracies along the way.

Another interesting observation in the training procedure is the rotation invariance of the data. When applying random rotation to the training data to make them "rotation invariant", the testing accuracy actually drops a lot. This is because "rotation invariant" may not present on the testing data. In common sense, food such as hamburgers, will have a piece of bread on the top, and another piece on the bottom. If it is rotated 90 degrees, the bread's position will make it "more" like a hotdog. Ice cream cones will not have pictures upside down. So, trying to make data rotation invariant does not make a lot of sense in this food dataset.

7 Team Contributions

Wenxiao Li: implemented data loaders, general procedures of training, validation and testing in engine.py. Built the baseline convolutional network from scratch. Applied transfer learning on Vgg 16. Wrote report sections of baseline, Vgg 16, and discussion.
Wenbo Hu: Implemented transfer learning, visulization of weights maps and feature maps on vgg16, resnet18 and custom model. Associatively helped general procedures of training and validation processes.
Huaning Liu: Implementation, optimization and experiment (transfer learning) for Resnet 18 and Custom Model, organization for latex report, wrote abstract, introduction, related work, and analysis for custom model and resnet 18 output.

References

[1] J. Ball, "Food classification with convolutional neural networks and multi-class linear discernment analysis," *CoRR*, vol. abs/2012.03170, 2020. [Online]. Available: https://arxiv.org/abs/2012.03170

[2] N. Martinel, G. L. Foresti, and C. Micheloni, "Wide-slice residual networks for food recognition," *CoRR*, vol. abs/1612.06543, 2016. [Online]. Available: http://arxiv.org/abs/1612.06543