STATS 415 Project Report Draft
Rotten food detection based on ResNet50

Zicong Xiao, Yulin Gao, Fengyuan Liu, Xiaoyang Sheng

December 9, 2022

1 Introduction

Neural networks have been rapidly developing in recent years and corresponding algorithms and structures have been applied to numerous fields in daily life. We specify our focus on one certain application, image classification, that is to identify the main objects from the images and classify them into several predefined groups. This technique can be used to develop an automatic rotten food detection system, which can be widely used from food production to consumption stages to reduce food waste, avoid possible economic loss, and prevent potential health threats. After reviewing several papers focusing on the task of rotten food detection, CNNs of ResNet50 is found to be the algorithm that achieves the highest average classification accuracy rate. For example, training on 2100 images of 6 classes of fresh/rotten fruits (i.e. orange, banana, and apple), the CNN-based rotten fruit detection model developed by C.C. Foong et.al.achieves a validation accuracy rate of 98.89 %. In this paper, we aim to reproduce the results obtained by C.C. Foong et.al. and further improve the proposed system by introducing an image augmentation technique to preprocess input images. Moreover, since the automatic rotten food detection technique developed by C.C. Foong et. al. only works for relatively small data sets with limited types of fruits (apple, banana, and orange) and it is likely that the classification accuracy of the proposed model will decrease when applying to other types of fruits or other foods. Thus, in this paper, we further proceed by training the model on a larger data set we found on the Kaggle website which contains 5997 images of 10 classes of rotten/fresh fruit (i.e. banana, mango, orange, apple, strawberry) and 10 classes of rotten/fresh vegetable (potato, cucumber, carrot, tomato, and bell pepper). Finally, we achieve the validation accuracy of 98.6% (normal transform), 96.7% (with the new augmentation method) over 6 classes and 97.7% (normal transform) over 20 classes.

2 Related Work

In recent years, there are a few research done on rotten fruit detection using different feature extraction techniques and classifiers. For example, using a dataset on Kaggle containing 1200 images of fresh/rotten apples, bananas, and oranges, Karakaya, D. et al. (2019) [1] compares the rotten fruit classification success rate of different feature extraction methods (such as Hist, GLCM, BoF, CNNsF) combined with binary SVMs classifiers. CNNs of a pre-trained ResNet-50 network is found to be the most successful feature extractor, combined with SVMs, it obtains an average success rate of 99.19% and 96.72 % for three- and six-class problems respectively.

In the work done by Ciocca, G. et al.(2018) [2], the researchers evaluate different CNN-based features learned from a CNN trained on a large food database with 475 food classes and 247,636 images. The classification accuracy of different CNN architectures (AlexNET, GoogleNet, VGGNet, and ResNet-50) was compared, and ResNet-50 appears to achieve the highest accuracy rate. In addition, the researchers achieved a large improvement in the accuracy of food image classifications when using a larger and more representative database.

The primary research that this paper aims to reproduce the results and make further improvements on is the one conducted by C. C. Foong et al. [3]. In this study, the researchers designed and developed intelligent rotten fruit detection to classify whether the apples, bananas, and oranges are fresh or rotten. The CNNs of ResNet50 was applied in this study for feature extractions and classification of rotten fruits. Three types of fruit (banana, orange, and orange) are detected and classified into rotten or fresh classes in this paper. The developed model achieved a validation accuracy of 98.89 %.

Convolutional Neutral Network (CNN) is one of the most popular deep neural networks. CNNs have been applied in many fields such as image recognition, image retrieval, image classification, and so on. CNN has an excellent performance in dealing with complex image inputs by reducing the number of parameters involved. CNNs

are comprised of three types of layers, which are convolutional layers, pooling layers, and fully-connected layers (Keiron O'Shea et al.) [4]. The input images will go through multiple layers, features in the images will be extracted during the process, and lastly using softmax function to classy input images based on the extracted features. The output of CNN is the output of the last fully connected layer, and the number of output nodes is equal to the number of image classes. Kernels are the main components in a CNN model coping with images. A (2m+1)*(2m+1), n kernel k can do the following convolution and produce results of n dimensions,

$$g(x,y) = k * f(x,y) = \sum_{dx=-m}^{m} \sum_{dy=-m}^{m} k(dx,dy) f(x-dx, y-dy)$$

where f is the original image and g is the one generated by convolution.

The first CNN architecture that has achieved better performance on the image classification task compared to the previous state-of-the-art is AlexNet (Hinton et al., 2012)[5]. Many other deeper architectures have been proposed since then, such as VGGNet, GoogleNet, and Residual Networks (ResNet). Among those architectures, the recognition accuracy achieved using ResNet won first place on the ILSVRC 2015 (ImageNet Large Scale Visual Recognition Challenge)(He et al., 2016)[6]. Residuals architectures are based on the idea that each layer of the network learns residual functions with reference to the layer's inputs instead of learning unreferenced functions. Residual networks were proved to be easier to optimize and to gain accuracy from considerably increased depth.

ResNet50 refers to the residual network of 50 layers. The architecture starts with one convolution layer involving 64 different kernels with the size of 7*7. The next convolution includes a 1*1, 64 kernel, a 3*3, 64 kernel, and a 1*1, 256 kernel, and repeats 3 times. The following convolution includes a 1*1, 128 kernel, a 3*3, 128 kernel, and a 1*1, 512 kernel, and repeat 4 times. The 4th convolution consists of three kernels of 1*1, 256, 3*3, 256, 1*1, 1024 and repeat 6 times. The 5th convolution, in addition, is made up of a 1*1, 512 kernel, a 3*3, 512 kernel and a 1*1, 2048 kernel and repeat three times. The architecture is ended with a fully connected layer containing 1000 nodes.

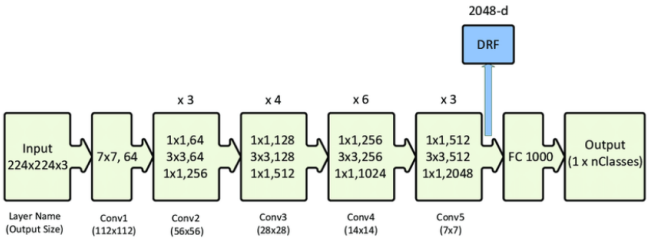

Figure 1: ResNet50 architecture [3]

3 Dataset and Features

3.1 Original Dataset of the Paper

There is one original dataset used by C.C. Foong et al.'s paper that we try to reproduce, which is available on Kaggle[7]. It contains roughly 500 images of 6 classes of rotten/fresh apple, banana, and orange. We follow the same procedure as the paper and use the data from the dataset in Table.1 .

	Fresh	Rotten
Apple	500	500
Banana	500	500
Orange	500	500

Table 1: The detailed information of the dataset from paper.

Figure.2 shows some examples of the dataset.

(a) Fresh Apple.

(b) Rotten Apple.

(c) Fresh Banana

(d) Rotten Banana.

Figure 2: Some examples of the paper's original dataset.

3.2 Upgraded Dataset

In addition to train our model on the original dataset used by C.C. Foong et al.'s paper, we found a larger dataset with 20 classes of rotten/fresh fruits and vegetables on the Kaggle Website[8].

This data set includes 5997 images of Fruits and 6003 images of Vegetables in total. Within Fruit/Vegetable category, there are 10 classes of 5 different types of fruit/vegetable, which are stored in the subfolders. The degree of rotten has many levels, some are slightly shrunk and some are even covered with a lot of molds. The detailed information of the dataset is shown in Table.2.

Fruit	Fresh	Rotten	Vegetable	Fresh	Rotten
Apple	612	588	Bellpeper	611	591
Banana	624	576	Carrot	620	580
Mango	605	593	Cucumber	608	593
Orange	609	591	Potato	615	585
Strawberry	603	596	Tomato	604	596

Table 2: The detailed information of the upgraded dataset.

As the description on the website, this dataset is gathered from different online sources, such as Google images, Bing images, Kaggle, Fruit360, and Sriram R.K., which provided samples of the pure-fresh category and a single item with a white background, respectively.

We use this upgraded dataset to further improve the model to apply the automatic rotten food detection algorithm on this expanded 20 classes.

Figure.3 are some examples of the images in the dataset.

(a) Fresh Apple.

(b) Rotten Apple.

(c) Fresh Tomato

(d) Rotten Tomato.

Figure 3: Some examples of the upgraded dataset.

3.3 Necessary Preprocessing of Data

Before we train the model, we need to preprocess the training data, which are the input fruit/vegetable images. This is because the input of the CNN model has to be the type of tensor. We use the **torchvision**, a package of **pytorch**, which is used for PyTorch Deep Learning, to build a computer vision model. We apply the transform functions to do the image preprocessing.

First, we resize the images to 256*256 and apply the central cropping for size 224. Since our model is based on ResNet, we need to transform the images to tensors, and normalize them to mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225].

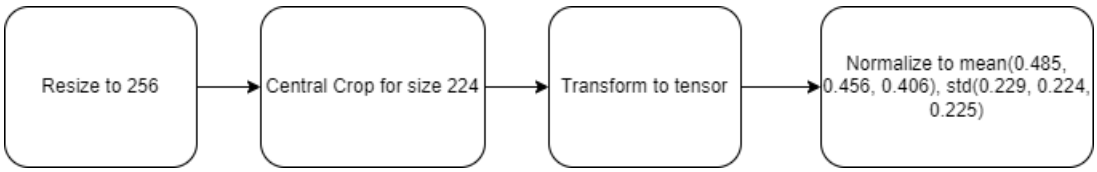

Figure 4: The procedure of transformation/preprocess.

Transformation above is a common procedure used by many algorithms and models of ResNet. We make some changes to this commonly used image preprocessing procedure to make it better fit our task.

4 Methods

4.1 Pre-fixed Model Configuration

There are some configurations and hyperparameters of CNN model that need to be set initially, and we choose some pre-fixed configurations that no further experimenting is required. The parameters set within the experiment will be explained in the following sections.

4.1.1 Train Test Split

The first configuration is to apply training and test dataset split. We pick 20% observations from the dataset as the validation set and leave the rest as the training set. Randomly generating the 20% indices, we shuffle them to generate the training and test indices. Training set and test set are then collected according to these indices.

4.1.2 Loss Function

To compare the model error of training set and test set, a shared loss function is required. We adopt CrossEntropyLoss() function in **torch.nn** package as our loss function. The function computes the cross entropy loss between input logits and target, with detailed documentation published at PyTorch's official website[9].

The input contains the unnormalized logits of each class and has to be a tensor of size C (the number of classes) with $K \geq 1$ for K-dimension case. The class indices are then in range of $[0, C)$, and the unreduced loss is[3]:

$$l(x, y) = L = l_1, ..., {l_N}^T, l_n = -w_{y_n} log \frac{exp(x_{n,y_n})}{\sum_{c=1}^{C} exp(x_{n,c})} \cdot 1y_n \neq \text{ignore\_index}$$

where x is input, y is a response, w is weight, C is number of classes, N spans the minibatch dimension as well as $d_1, ..., d_k$ for a high-dimensional case. If it is non-reduction, then[3]

$$l(x, y) = \begin{cases} \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n} \cdot 1y_n \neq \text{ignore\_index}} l_n, & \text{if reduction='mean'} \\ \sum_{n=1}^{N} l_n, & \text{if reduction='sum'} \end{cases} \tag{1}$$

4.1.3 Optimizer

Looking for an optimized model and an efficient way to reduce the loss in model, we use the torch.optim.SGD function as the optimizer during training process. It is a function in the package **torch.optim**. The detailed information and following algorithm are referred to PyTorch official website[10].

SGD is short for stochastic gradient decent, optionally with momentum. Corresponding algorithm is as below[4]:

input :γ(lr), θ_0(params), $f(\theta)$(objective), λ(weight decay), μ(momentum), τ(dampening), $nesterov, maximize$
for t=1 **do**
  $g_t \leftarrow \nabla_\theta f_t(\theta +_{t-1})$
  if $\lambda \neq 0$ **then**
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  end if
  if $\mu \neq 0$ **then**
    if $t > 1$ **then**

$b_t \leftarrow \mu b_{t-1} + (1-\tau) g_t$
else
$b_t \leftarrow g_t$
end if
if *nesterov* **then**
$g_t \leftarrow g_t + \mu b_t$
else
$g_t \leftarrow b_t$
end if
end if
if *maximize* **then**
$\theta_t \leftarrow \theta_{t-1} + \gamma g_t$
else
$\theta_t \leftarrow \theta_{t-1} - \gamma g_t$
end if
end for
return θ_t

In our model, $\gamma = 0.001, \mu = 0.9$, others as default, e.g. $\lambda = 0$, $\tau = 0$, $nesterov = False$, $maximize = False$.

4.1.4 Modification on Fully Connected Layer

Since a CNN that has been trained for solving a given task can also be adapted to solve a different task and it is hard to find a sufficient data set to train an entire CNN from scratch (Ciocca, G. et al.,2018) [2], a pre-trained CNN ResNet50 in **torch** package in Python is used in this paper. The pretrained ResNet50 could be used to classify images into 100 classes[5], which means the pretrained networks have been learning a lot of features of the images. We further modify this pretrained networks with transfer learning to make it fit our scenario. The modifications we did on the pretrained algorithm are as follows:

We replace the last fully connected layer (FC layer) with a new FC layer, making the number of outputs corresponding to the number of classes that we try to classify. In our case, there are 6 classes (fresh apple, rotten apple, fresh orange, rotten orange, fresh banana, rotten banana). Then we train our model for several epochs to calculate and minimize the loss function and optimize the model by the pre-defined optimizer algorithm. Here some hyperparameters are involved and matter, such as the number of epochs and batch size, we later did comparison experiment on these hyperparameters and get a serial of output and validation error, which are discussed in the Experiment section.

4.2 Image Augmentation Experiment

Deep neural networks usually require a relatively large dataset to develop good models. A limited dataset, however, can cause overfitting problems and even lead to a biased model. One issue with the current dataset is that many images to be used for training are highly similar to each other and even repeat in the set. In Figure. 5, for example, the second and the third images are almost "the same". Considering it's source-consuming to collect more images and label them, image augmentation is a good solution instead. Image augmentation is a process of creating new training samples by applying transformations to existing ones. This practice can enlarge the sets but add to the diversity between similar samples as well.

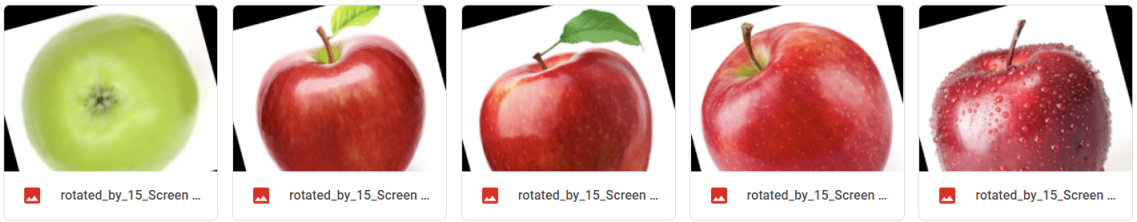

Figure 5: similar images in training set

Five image augmentation methods are introduced in this experiment: cropping, flipping, color jittering, random erasing, and sharpening. Image cropping is trimming off the edges of the original image with the center fixed, making the network only focus on part of an image during the training process. Image flipping is flipping the image in the left-right direction or the up-down direction. As a more general case, image rotation turns the original image by a random degree. Such augmentation techniques expand the training sets without getting rid of any

information. The remaining three techniques, in contrast, remove some features of the original image to avoid overfitting. Color jittering randomly changes the values of exposure and saturation. Random erasing randomly selects a rectangle region and erases the pixels. In addition, sharpening randomly increases the sharpness of an image. These techniques strengthen the trained model to be robust to images of various qualities.

Figure 6: Original

Figure 7: Crop

Figure 8: Flip

Figure 9: Color jitter

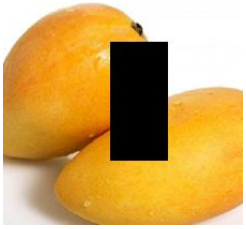

Figure 10: Random erase

Figure 11: Sharpen

Five experiments are conducted with different image augmentation combinations. In all five experiments, we fix epoch at 6 and batch size at 30. The highest accuracy is achieved as 98.8% in experiment 1, when only center cropping is added to preprocessing transformation. The lowest accuracy decreases to 94.0% in experiment 3 when color jittering is adopted as an augmentation technique. The model accuracy is high on average in the image augmentation experiments, so the model performs well in avoiding overfitting problems.

Experiment	Description	Accuracy(%)
1	crop to 224*224	98.8
2	crop; add random flip & rotation	97.7
3	crop; add random flip & rotation; add color jitter	94.0
4	crop; add random flip & rotation; add random erase	96.2
5	crop; add random flip & rotation; add sharpening	96.7

4.3 Other Parameters Experiment

In this part, we will introduce the experiment we did on some hyperparameters and the corresponding validation error obtained, including batch size and epoch.

4.3.1 Experiment Setup

Before we start the experiment, there is some setup. The initial learning rate is set as 0.0001, the default mini-batch size is set as 10, and the default epoch size is 6. We try to investigate the effect of different mini-batch sizes and the number of epochs. We test the mini-batch size as (5,10,50,100) and the epoch size as (3,6).

4.3.2 Experiment Result

First, in the below table, we did not apply the image augmentation experiment, only the necessary preprocessing.

Epoch size(under batch-size=10)	Test accuracy
Inference without training	18.1%
Inference after training 1 epoch	90.9%
Inference after training 3 epoch	95.4%
Inference after training 6 epoch	98.6%

Batch size(under epoch=6)	Test accuracy
5	98.8%
10	98.6%
50	94.2%
100	92.2%

4.3.3 Experiment Discussion and adding image augmentation

From the table in the result section, we finally pick epoch = 6 and batch size = 10. And then we use this parameter configuration to apply the image augmentation and obtain the test accuracy as 96.7%, which is a little smaller than 98.6% with the original preprocessing.

We explain that the validation accuracy we obtained here is after applying the image augmentation technique, which avoids the problem of overfitting on this particular dataset by adding diversity to the input images. Though the test accuracy is a little bit smaller, our model may achieve better performance on other datasets.

4.4 Experiment with the Upgraded Dataset

After successfully reproducing the model using the original dataset, we carried out several experiments using the upgraded dataset and here are the results.

4.4.1 Binary Classification of Rotten Fruit/Vegetables

For each kind of fruit or vegetables in the upgraded dataset, we perform a train loop and a validation, with batch size 10 and epoch size 10. The validation accuracy of each category is shown below:

Category	Validation Accuracy	Category	Validation Accuracy
Apple	97.8%	Banana	99.7%
Mango	99.7%	Orange	97.8%
Strawberry	99.2%	Bell Peper	96.1%
Carrot	95.3%	Cucumber	98.6%
Patato	95.3%	Tomato	99.4%

4.4.2 Rotten/Fresh Detection of 5 kinds of fruits and 5 kinds of vegetables

In this experiment, we perform a 20-class classification using the upgraded dataset, which contains 5 types of fruits and 5 types of vegetables, each kind of which will be predicted as rotten or fresh.

We set the default batch size as 10 and the default epoch size as 6. The validation accuracy achieved is **95.6%**.

4.5 Binary Classification Experiment across Two Datasets

In this section, we try to check the performance of the binary detection (only output fresh or rotten) of the model trained in one dataset on the other dataset as validation test. Here are the results:

- First, we perform classification on the original dataset, the validation accuracy is 98.6%.
- Second, we perform classification on the upgraded dataset, the validation accuracy is 97.7%.
- Third, we train on the upgraded dataset and test on the original dataset without the refined augmentation methods. The test accuracy without augmentation is 91.5%, and accuracy with augmentation is 92.8%.
- Last, we train on the original dataset and test on the upgraded dataset with and without the refined augmentation methods. The test accuracy without augmentation in the training part is 79.7%, the test accuracy with augmentation is 77.8%.

We can see that though there are significant difference of two dataset, the validation accuracy is acceptable, except the model trained on the original data and tested on the upgraded one. Therefore, we may still train the model on the upgraded dataset.

5 Result and Discussion

After experiment and configuration in the reproducing of the paper model, we achieved a test accuracy of **98.6%** (normal transform) and **96.7%** (with the new augmentation method) over 6 classes of rotten/fresh fruits (rotten/fresh orange, banana, apple). This is close to the original results obtained by C.C. Foong's paper. Moreover, we introduce a new transform-preprocessing method to fix the problems of the high similarity of some images in the dataset, which successfully solve the overfitting problem.

Then we enlarge the training data set to more kinds of fruits and vegetables, which are 20 classes in total (rotten/fresh apple, banana, mango, orange, strawberry, bell pepper, carrot, cucumber, potato and tomato). The validation accuracy is **97.7%** (normal transform), which is still a high accuracy rate, and therefore the model is workable for other types of fruits and vegetables.

Although the newly proposed image augmentation method help solve the overfitting issue, it reduces the test accuracy. We may further improve the method in the future so both of them could be achieved. Besides, the proposed model may tend to depend too much on the color of the fruits and vegetables due to the characteristics of the training data. We did some experiments after applying the gray processing to the data, the accuracy drops a little bit, which is around 85%. Therefore, there are still improvements on the color-dependency of the model and we look forward to fixing this issue in the future.

6 Code Work

All the codes developed and applied in this project is uploaded to the Github, the link is: `https://github.com/gao-yulin/rotten-detect-resnet`.

7 Acknowledgement

Here are the details of the distribution of this project:

Yulin Gao is responsible for implementing the Resnet50 network for the supervised classification algorithm, and performs various experiments to fix the optimal hyperparameter and test the performance of the model across two specified datasets.

Xiaoyang Sheng is responsible for the neural network tryout, writing and learning of the details of theoretical knowledge and information of the methods applied, and the writing of the dataset, method, conclusion part of the report.

Zicong Xiao is responsible for conducting the image augmentation experiment to check overfitting issue, collect expanded dataset, modify the final report.

Fengyuan Liu is responsible for putting every group member's inputs together, collecting related information and materials for our project, studying and writing theroetical concepts, and revising our final report.

8 Reference

[1] D. Karakaya, O. Ulucan and M. Turkan, "A Comparative Analysis on Fruit Freshness Classification," 2019 Innovations in Intelligent Systems and Applications Conference (ASYU), 2019, pp. 1-4, doi: 10.1109/ASYU48272.2019.8946385.
[2] Ciocca, G., Napoletano, P., Schettini, R. (2018). CNN-based features for retrieval and classification of food images. Computer Vision and Image Understanding, 176, 70-77.
[3] C. C. Foong, G. K. Meng and L. L. Tze, "Convolutional Neural Network based Rotten Fruit Detection using ResNet50," 2021 IEEE 12th Control and System Graduate Research Colloquium (ICSGRC), 2021, pp. 75-80, doi: 10.1109/ICSGRC53186.2021.9515280.
[4] O'Shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." arXiv preprint arXiv:1511.08458 (2015).
[5]Hinton, Geoffrey, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." IEEE Signal processing magazine 29, no. 6 (2012): 82-97.
[6]He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.
[7] K. R. Sriram, "Kaggle - Fruits fresh and rotten for classification," 2018. [Online]. Available: `https://www.kaggle.com/sriramr/fruits-fresh-and-rotten-for-classification`.
[8] M. Mukhriddin, "Kaggle - Fruits and Vegetables dataset," 2022. [Online] `https://www.kaggle.com/datasets/muhriddinmuxiddinov/fruits-and-vegetables-dataset`.
[9] "PyTorch - CROSSENTROPYLOSS," 2022. [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`.
[10] "PyTorch - SGD," 2022. [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.optim.SGD.html`.