1st World Congress on Industrial Engineering and Operations Management
October 9-11, 2024
Publisher: IEOM Society International, USA DOI: 10.46254/WC01.20240170
Published: October 09, 2024,

A Study on the Method of Classifying Spoiled Fruits Using Convolutional Neural Networks (CNN)

Yuchan Lee, Shin Dong Ho
Student and Professor, My Paul School
12-11, Dowontongmi-gil, Cheongcheon-myeon, Goesan-gun
Chungcheongbuk-do, Republic of Korea
eavatar@hanmail.net

Abstract

This study proposes a method for determining the freshness of fruits using a Convolutional Neural Network (CNN) model. Spoiled fruits often exhibit minimal visual changes, making them difficult to distinguish with the naked eye. To address this, an algorithm was developed that utilizes a CNN model to automatically learn features such as color, texture, and shape of the fruits, and determine their freshness. The experiment utilized image datasets of both fresh and spoiled fruits, including various types such as apples and oranges. The structure of the CNN model consists of convolutional layers, pooling layers, and fully connected layers, designed to determine the freshness of the fruits in the final output. The experimental results demonstrated that the CNN model could identify spoiled fruits with an accuracy of over 92%. This indicates that the proposed method provides an objective and accurate way to determine fruit freshness compared to traditional visual inspection methods. The suggested technology is expected to be applicable in various fields such as fruit quality control and distribution process monitoring.

Keywords
CNN, Convolutional Neural Network, Classifying Spoiled Fruits, fruits and AI

1. Introduction
Ensuring the supply of fresh fruit in the food industry presents a significant challenge. Spoiled fruit not only poses a risk to consumer health but also results in substantial economic losses. Key factors contributing to fruit spoilage during distribution include inadequate post-harvest handling, insufficient temperature and humidity control, and prolonged storage periods. Consequently, to mitigate these issues and provide consumers with fresh fruit, there exists a critical need for technologies that can efficiently and accurately detect and sort spoiled fruit during distribution. Conventional methods of assessing fruit freshness, such as visual inspection and rudimentary sensor-based approaches, have been demonstrated to lack both accuracy and efficiency. Recent advancements in computer vision and deep learning have facilitated the automated learning and analysis of fruit appearance. Specifically, Convolutional Neural Networks (CNNs) have demonstrated efficacy in extracting features from image data, rendering them highly suitable for assessing fruit freshness. This investigation aims to develop a system utilizing CNN models to reliably determine fruit freshness (i.e., fresh or spoiled) and achieve an accuracy exceeding 95% in the identification of spoiled fruit. By advancing technology for the automated detection of spoiled fruit during distribution, this study endeavors to reduce food waste and enhance the preservation of freshness throughout the supply chain.

2. Body
2.1 Application of CNN for Spoiled Fruit Classification
To implement the model for detecting spoiled fruit, Keras and OpenCV were utilized. Keras is a high-level neural network API designed for deep neural network modeling, supporting various backend engines such as TensorFlow, MATLAB, and Theano, and is an open-source deep learning library. For this investigation, TensorFlow was employed as the backend.

OpenCV (Open Source Computer Vision Library) is an open-source library designed for the development of real-time computer vision and machine learning applications. It supports a wide range of programming languages (C++, Python, Java, etc.) and provides an extensive collection of algorithms and functions. This library can be applied in diverse fields such as image processing, video analysis, object detection, facial recognition, and the integration of machine learning models.

2.2 Model Design

The model architecture is structured as follows:
Convolutional Layers
- **First Convolutional Layer** utilizing `Conv2D`: 32 filters of size 3x3 are employed to extract initial features from the input image. The activation function utilized is ReLU (Rectified Linear Unit).
- **First Pooling Layer** utilizing `MaxPooling2D`: 2x2 max pooling is implemented to reduce the size of the feature maps and emphasize salient features.
- **Second Convolutional Layer** utilizing `Conv2D`: 64 filters of size 3x3 are applied to extract more complex features.
- **Second Pooling Layer** utilizing `MaxPooling2D`: 2x2 max pooling is applied again to further reduce the size of the feature maps.
- **Third Convolutional Layer** utilizing `Conv2D`: 128 filters of size 3x3 are employed to extract higher-level features.
Global Average Pooling Layer
- This layer computes the average of each feature map, reducing the dimensionality and preparing the data for the fully connected layer.
Fully Connected Layers
- **Dense Layer**: A fully connected layer with 128 neurons is utilized to learn the extracted features. The activation function is ReLU.
- **Dropout Layer**: A dropout layer is implemented to randomly exclude certain neurons during training to mitigate overfitting.
- **Output Layer** utilizing `Dense`: A softmax activation function is employed to calculate the probability for each class.

A Convolutional Neural Network (CNN) is a representative deep-learning model known for its exceptional performance in image classification and object detection. The structure of the CNN is as follows: The Input Layer received the fruit image data to be classified. The Convolutional Layer is the core component that extracts features from an image by detecting local patterns using the learned filters. The Activation Layer introduces nonlinearity by applying the ReLU (Rectified Linear Unit) function, which enhances the representational capacity of the model. The Pooling Layer reduces the size of the feature maps, decreases the computational load, and helps prevent overfitting. Typically, max-pooling or average-pooling is used. The Fully Connected Layer integrates the extracted features to perform final classification. The training process of a CNN model proceeds as follows. First, the model is initialized with random weights and biases, and a loss function is defined. The loss function measures the difference between the predicted output of the model and actual labels. The model parameters were updated to minimize this loss. By repeating this process, the model learns to effectively capture the features of the fruit images.

3 Model Implementation & Discussions

Figure 1. Google Drive Mount

This image shows the process of mounting Google Drive in the Google Colab environment. The drive module from Colab was imported using the Google Colab import drive, and the command drive.mount('/content/drive') was

executed to connect Google Drive to Colab. Once this command is executed, the user is prompted to authenticate, thereby allowing access to Google Drive. After successful authentication, the files stored in Google Drive can be accessed and utilized within the Colab environment.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator


# Image Data Path Configuration
train_dir = '/content/drive/MyDrive/train'
validation_dir = '/content/drive/MyDrive/test'
```

Figure 2. Importing Libraries and Setting Image Data Path

This image presents the code for setting the paths for TensorFlow libraries and image data processing using Google Colab. The code imports TensorFlow and ImageDataGenerator to facilitate preprocessing of the image data. The train_dir variable points to the path where the training data are stored, specifically by referencing the train folder in Google Drive. The validation_dir variable indicates the path for the validation data pointing to the test folder in Google Drive. By specifying the folders containing the image data using this code, the designated paths can be utilized during the training and validation processes.

```
# Data Augmentation and Preprocessing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)
```

```
Found 9695 images belonging to 2 classes.
Found 2399 images belonging to 2 classes.
```

Figure 3. Data Augmentation and Preprocessing

Keras's ImageDataGenerator was utilized to train the image classification model. The train_datagen and validation_datagen were configured to load and preprocess the training data from train_dir and validation data from validation_dir through image augmentation. The training data undergo various transformations, including rotation, translation, zooming, and horizontal flipping, and are normalized (rescaled) to make them suitable for the model. The train_generator and validation_generator loaded the training and validation data in batch units, respectively, and were structured to perform binary classification (with class_mode='binary').

```
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
```

Figure 4. Model Definition

A Convolutional Neural Network (CNN) model was defined using Keras. A Sequential model was employed to stack layers sequentially. The first Conv2D layer applies 32 filters, with the input image size defined as (150, 150, 3), indicating that it accepts RGB image data with a size of 150 × 150 pixels. Each Conv2D layer utilizes a 3 × 3 kernel and incorporates the rectified linear unit (ReLU) activation function to introduce nonlinearity. Following each convolutional layer, a MaxPooling2D layer is added to reduce the spatial dimensions. The number of filters was gradually increased to 64 and then 128, thereby deepening the network. A flattened layer was utilized to convert the 2D output into a 1D format, followed by the addition of a fully connected (dense) layer. The final dense layer employs the sigmoid activation function for binary classification, outputting values between zero and one. The model was compiled using the binary cross-entropy loss function and Adam optimization algorithm, with accuracy as the performance metric.

```
history = model.fit(
    train_generator,
    steps_per_epoch=100,  # When the batch size is 20, 100 steps are required to process 2000 images.
    epochs=30,
    validation_data=validation_generator,
    validation_steps=50  # When the batch size is 20, 50 steps are required to process 1000 images.
)
```

Figure 5. model learning

The Keras model was trained using a fitting method. The train_generator supplied the training data in batches. The parameter steps_per_epoch=100 indicate that with a batch size of 20, the model processes 2000 images during one epoch, requiring 100 steps. Parameter epochs=30 specify that the model will be trained for a total of 30 epochs. The validation data were provided through the validation_generator, and the number of steps for validation was set to validation_steps=50, which corresponds to processing 1000 images with a batch size of 20. In summary, this code conducts training over 30 epochs, using 2000 training images for each epoch, and evaluates the performance with 1000 validation images.

```
1 loss, accuracy = model.evaluate(validation_generator)
2 print(f'Validation Loss: {loss}')
3 print(f'Validation Accuracy: {accuracy}')
```

```
120/120 [==============================] - 54s 448ms/step - loss: 0.2766 - accuracy: 0.9254
Validation Loss: 0.27662762999534607
Validation Accuracy: 0.925385594367981
```

Figure 6. Model Evaluation

The model was evaluated using model evaluation (validation_generator), which calculates the loss and accuracy, and the results were subsequently printed. Evaluation Results: Validation Loss: 0.2766 Validation Accuracy: 0.9254 Therefore, the model achieved an accuracy of 92.54% for the validation data, with a loss value of approximately 0.2766.

4. Conclusion

Research Results

This study developed a system for effectively determining the condition of fruits using a Convolutional Neural Network (CNN) model. The experimental results indicate that the CNN model can distinguish between fresh and spoiled fruits with an average accuracy exceeding 92%. This demonstrates that the CNN effectively learns the visual characteristics of fruits and can accurately assess their condition based on this learning. However, the study did not achieve a target accuracy of 95%, and the intended use of OpenCV in the model design was not implemented.

Limitations and Future Plans

The accuracy of the model can be improved by reconstructing it using OpenCV and augmenting the training data.

References

Arya R., Khanduja M., Rani, P. S., Pundhir, P., Tiwari, M., and Shelke, C. J., Empirical Analysis of Deep Learning for Big Data and Its Applications using CNN. *In 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. pp. 01-05. IEEE. 2022.

Song J., Gao S., Zhu Y., and Ma C., A survey of remote sensing image classification based on CNNs. *Big earth data*, 3(3), pp.232-254, 2019.

Rahman M., Prodhan R., Shishir Y., and Ripon S., Analyzing and evaluating boosting-based CNN algorithmsfor image classification. *In 2021 International Conference on Intelligent Technologies (CONIT)*, pp. 1-6. IEEE, 2021.

Pu S., Olshevsky A., Paschalidis I. Ch., Asymptotic Network Independence in Distributed Stochastic Optimization for Machine Learning: Examining Distributed and Centralized Stochastic Gradient Descent, pp.114 – 122, *IEEE signal processing magazine*, 2020

Zavala M., Luis A., Lamichhane B., Zhang L., Haan G., CNN-SkelPose: a CNN-based skeleton estimation algorithm for clinical applications, *Journal of Ambient Intelligence and Humanized Computing*, pp.2369 – 2380, 2020

Chin T. L., Chao H. H., Shi A. C., CNN-Based Hybrid-Order Texture Segregation as Early Vision Processing and Its Implementation on CNN-UM, pp.2277 – 2287, *IEEE*, 2007

Vladimirov N., Brui E., Levchuk A., Al H., Walid F., Vladimir E., Aleksandr B., David, CNN-based fully automatic wrist cartilage volume quantification in MR images: A comparative analysis between different CNN architectures, pp.737 – 751, *Magnetic resonance in medicine: official journal of the Society of Magnetic Resonance in Medicine*, 2023

Dai Y., Jin I., Song Y., Du H., Zhao D., CNN-based multiple-input multiple-output radar image enhancement method, Dai, pp.6840 – 6844, *JoE*, 2019

Arena P., Basile A. , Fortuna L. , Mazzitelli G., Rizzo A. , Zammataro M., CNN-L4.5: cnn-based real-time video detection of plasma instability in nuclear fusion applications, pp.77 – 80, *IEEE*, 2004

Mandolesi P. S., Julian P., Andreou A. G., CNN-L2.3: A simplicial cnn architecture for on-chip image processing, pp.29 – 32, *IEEE*, 2004

paasio a., flak j., laiho m., halonen k., cnn-l2.1: high density vlsi implementation of a bipolar cnn with reduced programmability, pp.21 – 24, *ieee*, 2004

yang z., tsuruta k., nishio y., ushida a., cnn-l3.3: investigation of phase-wave propagation phenomena in second order cnn arrays, pp.49 – 52, *ieee*, 2004

balya d., petras i., rekeczky c., cnn-L3.4: pattern formation on the prototype complex-cell cnn-um chip (cace1k), Balya, pp.53 – 56, *IEEE International Symposium on Circuits and Systems*, 2004

Shi B., Luo T., CNN-L3.5: A 32 x 32 four layer reaction-diffusion cnn chip, pp.57 – 60, *IEEE International Symposium on Circuits and Systems*, 2004

Cserey G., Falus A., Porod W., Roska T., CNN-L1.5: AN ARTIFICIAL IMMUNE SYSTEM FOR VISUAL APPLICATIONS WITH CNN-UM, pp.17 – 20, *IEEE International Symposium on Circuits and Systems*, 2004

Biographies

Yuchan Lee is student in MY PAUL SCHOOL. He is interested in artificial intelligence, deep learning, cryptography, robots, autonomous vehicles, etc., and is conducting related research.

Shin Dong Ho is Professor and Teacher in MY PAUL SCHOOL. He obtained his Ph.D. in semiconductor physics in 2000. He is interested in artificial intelligence, deep learning, cryptography, robots, block chains, drones, autonomous vehicles, mechanical engineering, the Internet of Things, metaverse, virtual reality, and space science, and is conducting related research.