International Journal of Research in Engineering, Science and Management
Volume 4, Issue 11, November 2021
https://www.ijresm.com | ISSN (Online): 2581-5792

Fruit Quality Classification Using CNN

Nandila Bhattacharjee*
Student, Department of Computer Science Engineering, SRM Institute of Science and Technology, Chennai, India

***Abstract*: Fruit Classification has become a riveting topic in computer vision. Traditional fruit classification processes are generally based on visual ability and such methods can be very tedious, inconsistent and time consuming. In agriculture science, fruit classification can be found highly beneficial. Hence researches in this area indicate the feasibility of using deep learning models to improve product quality while liberating people from the traditional hand sorting of fruits. In this project an extensive dataset of 3 varieties of fruits is considered. With this dataset, an ImageNet pre-trained convolutional neural network was fine-tuned to obtain a classifier. This classifier is optimized to obtain high accuracy in less time for the classification of rotten and fresh fruits.**

***Keywords*: CNN, Python, Transfer learning, Deep learning, Tensorflow, Keras.**

1. Proposed System

The novelty of my work is to provide a classification system which can classify rotten and fresh fruits in less time with very high efficiency.

2. Existing System

Hyperspectral fruit and vegetable classification using convolutional neural networks, Jan Steinbrener, Konstantin Posch, Raimund Leitner.

In this paper, they show an easy way to classify hyperspectral images with state of the art convolutional neural networks pre-trained for RGB image data.

Fruit Image Classification Using Convolutional Neural Networks, Shawon Ashraf, Ivan Kadery, Md Abdul Ahad Chowdhury, Tahsin Zahin Mahbub and Rashedur M. Rahman.

They built the initial model using the Inception V3 model and trained with our dataset applying transfer learning to predict if the fruit is rotten or fresh in the given dataset

3. The Dataset

Fig. 1. Image samples from the dataset

The dataset comes from Kaggle, there are 6 categories of fruits: fresh apples, fresh oranges, fresh bananas, rotten apples, rotten oranges, and rotten bananas.10904 images for testing data and 2698 images for training data.

4. Algorithms and Concepts Used

A. Convolution Neural Network (CNN)

CNN is also known as feed forward neural network which is mainly used for image classification. They are often upto 20-30 layers which include input layer, dense layer , output layer with activation function etc.

B. ImageNet

ImageNet7 is a large scale image database. ImageNet as a foundation when applied in training techniques on CNN they categories the database provides.

C. Keras

Keras is an open-source neural-network library written in Python, which is used for the evaluating, preprocessing, modeling, and optimization. It is capable of running on top of TensorFlow.

D. Softmax Activation Function

The softmax function is used as the activation function in the output layer of neural network models that predict a multinomial probability distribution. That is is used as the activation function for multi-class classification problems where class membership is required on more than two class labels.

E. Project Layout

Steps followed are:

Step 1: Loading ImageNet Base Model

The ImageNet pre-trained models are often good choices for computer vision transfer learning, as they have learned to classify various different types of images. So we start by downloading the pre-trained model. This is available directly from the Keras library

Step 2: Freeze Base Model

This is done so that all the learning from the ImageNet dataset does not get destroyed in the initial training.

Step 3: Add Layers to Model

Then we add layers to the pretrained model

Step 4: Compile Model

*Corresponding author: nandila2711@gmail.com

Compilation is done using compile function with categorical cross entropy and adams optimizer.

Step 5: Augment the Data

Keras comes with an image augmentation class called Image Data Generator

Step 6: Load Dataset

Mounted on google drive then we are going to load images directly from folders using Keras' flow_from_directory function.

Step 7: Train the Model

Pass the train and valid iterators into the fit function, as well as setting your desired number of epochs.

Step 8: Unfreeze Model for Fine Tuning

We unfreeze the entire model and train it again with a very small learning rate. This will cause the base pre-trained layers to take very small steps and adjust slightly, improving the model by a small amount.

Step 9: Evaluate the Model

The evaluate function will return a tuple, where the first value is your loss, and the second value is your accuracy.

F. Code

```
from tensorflow import keras
base_model = keras.applications.VGG16(
weights='imagenet', # Load weights pre-trained on ImageNet.
    input_shape=(224, 224, 3),
    include_top=False)
base_model.trainable = False
inputs = keras.Input(shape=(224, 224, 3))
x = base_model(inputs, training=False)
# pooling layer or flatten layer
x = keras.layers.GlobalAveragePooling2D()(x)
# Add final dense layer
outputs = keras.layers.Dense(6, activation = 'softmax')(x)
model = keras.Model(inputs, outputs)
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
from
tensorflow.keras.preprocessing.imageimport ImageDataGenerator
datagen = ImageDataGenerator(samplewise_center=True,
# set each sample mean to 0
rotation_range=10,
# randomly rotate images in the range (degrees, 0 to 180)
zoom_range = 0.1, # Randomly zoom image
width_shift_range=0.1, # randomly shift images horizontally (fraction of total width)
height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
horizontal_flip=True, # randomly flip images
vertical_flip=False) # we don't expect fruit to be upside-down so we will not flip vertically
# load and iterate training dataset
train_it= datagen.flow_from_directory('/content/drive/MyDrive/dataset/train',
                                      target_size=(224, 224),
                                      color_mode='rgb',
                                      class_mode="categorical")
# load and iterate validation dataset
valid_it = datagen.flow_from_directory('/content/drive/MyDrive/dataset/test',
                                       target_size=(224, 224),
                                       color_mode='rgb',
                                       class_mode="categorical")
model.fit(train_it,
      validation_data=valid_it,
      steps_per_epoch=train_it.samples/train_it.batch_size,
      validation_steps=valid_it.samples/valid_it.batch_size,
      epochs=5)
# Unfreeze the base model
base_model.trainable = True
# Compile the model with a low learning rate
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate = .00001),
loss= keras.losses.CategoricalCrossentropy(from_logits=True) , metrics =[keras.metrics.CategoricalAccuracy()] )
model.fit(train_it,
      validation_data=valid_it,
      steps_per_epoch=train_it.samples/train_it.batch_size,
      validation_steps=valid_it.samples/valid_it.batch_size,
      epochs=5)
model.evaluate(valid_it, steps=valid_it.samples/valid_it.batch_size)
```

G. Code Screenshots and Explanation

Loading ImageNet Base Model

```
[ ]  from tensorflow import keras

     base_model = keras.applications.VGG16(
         weights='imagenet',  # Load weights pre-trained on ImageNet.
         input_shape=(224, 224, 3),
         include_top=False)
```

Fig. 2. Loading ImageNet base model

The ImageNet pre-trained models are often good choices for computer vision transfer learning, as they have learned to classify various different types of images. In doing this, they have learned to detect many different types of features that could be valuable in image recognition. Hence the first step is downloading the pre-trained model. This is available directly from the Keras library. As we are downloading, there is going to be an important difference. The last layer of an ImageNet model is a dense layer of 1000 units, representing the 1000 possible classes in the dataset. In our case, we want it to make a different classification: is this rotten or not? Because we want the classification to be different, we are going to remove the last layer of the model. We can do this by setting the flag include_top=False when downloading the model. After removing this top layer, we can add new layers that will yield the type of classification that we want:

Freeze Base Model

```
[ ]  base_model.trainable = False
```

Fig. 3. Freeze base model

This is done so that all the learning from the ImageNet dataset does not get destroyed in the initial training.

Freezing the model's pre-trained layers. This means that when we train, we will not update the base layers from the pre-trained model. Instead we will only update the new layers that we add on the end for our new classification. We freeze the initial layers because we want to retain the learning achieved from training on the ImageNet dataset. If they were unfrozen at this stage, we would likely destroy this valuable information. There will be an option to unfreeze and train these layers later, in a process called fine-tuning. Freezing the base layers is as simple as setting trainable on the model to False.

Add Layers to Model

```
[ ]  inputs =  keras.Input(shape=(224, 224, 3))

     x = base_model(inputs, training=False)

     # pooling layer or flatten layer

     x = keras.layers.GlobalAveragePooling2D()(x)

     # Add final dense layer
     outputs = keras.layers.Dense(6, activation = 'softmax')(x)

     model = keras.Model(inputs, outputs)
```

Fig. 4. Add layers to model

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 6)                 3078
=================================================================
Total params: 14,717,766
Trainable params: 3,078
Non-trainable params: 14,714,688
_________________________________________________________________
```

Fig. 5. Model summary

The addition of a pooling layer after the convolutional layer is a common pattern used for ordering layers within a convolutional neural network that may be repeated one or more times in a given model. The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.

The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps. Most of the time the Softmax Function is related to the Cross Entropy Function.

Compile Model

```
[ ]  model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
```

Fig. 6. Compile model

The need for cross entropy is so in working out the cross entropies of each observation shows that when the model incorrectly predicted 1 with a low probability, there was a smaller loss than when the model incorrectly predicted 0 with a high probability. Minimizing this loss function will prevent high probabilities from being assigned to incorrect predictions.

Adam Optimizer Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data

Above the metric accuracy is Number of correct predictions.

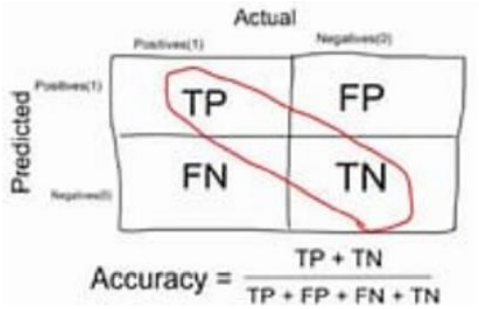

Fig. 7. Accuracy

Augment the Data

```
[ ]  from tensorflow.keras.preprocessing.image import ImageDataGenerator

     datagen = ImageDataGenerator(samplewise_center=True,  # set each sample mean to 0
             rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
             zoom_range = 0.1, # Randomly zoom image
             width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
             height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
             horizontal_flip=True,  # randomly flip images
             vertical_flip=False) # we don't expect fruit to be upside-down so we will not flip vertically
```

Fig. 8. Augment the data

Keras comes with an image augmentation class called Image Data Generator which is used to allow the model to see a wider variety of images to learn from. This will help it learn to recognize new pictures of fruits instead of just memorizing the pictures it trains on.

Load Dataset

```
[ ]  # load and iterate training dataset
     train_it = datagen.flow_from_directory('/content/drive/MyDrive/dataset/train',
                                            target_size=(224, 224),
                                            color_mode='rgb',
                                            class_mode="categorical")
     # load and iterate validation dataset
     valid_it = datagen.flow_from_directory('/content/drive/MyDrive/dataset/test',
                                           target_size=(224, 224),
                                           color_mode='rgb',
                                           class_mode="categorical")

     Found 10904 images belonging to 6 classes.
     Found 2698 images belonging to 6 classes.
```

Fig. 9. Load dataset

Mounted on google drive then we are going to load images directly from folders using Keras' flow_from_directory function.

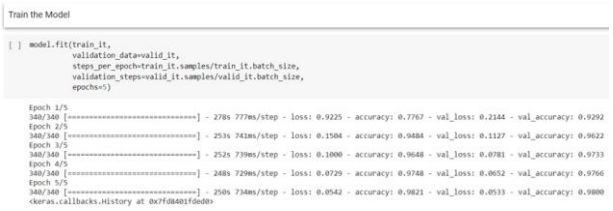

Fig. 10. Train the model

Pass the train and valid iterators into the fit function, as well as setting your desired number of epochs.

Fig. 11. Unfreeze model for fine tuning

Unfreeze the entire model and train it again with a very small learning rate. This will cause the base pre-trained layers to take very small steps and adjust slightly, improving the model by a small amount. Note that it is important to only do this step after the model with frozen layers has been fully trained. The untrained pooling and classification layers that we added to the model earlier were randomly initialized. This means they needed to be updated quite a lot to correctly classify the images. Through the process of backpropagation, large initial updates in the last layers would have caused potentially large updates in the pre-trained layers as well. These updates would have destroyed those important pre-trained features. However, now that those final layers are trained and have converged, any updates to the model as a whole will be much smaller (especially with a very small learning rate) and will not destroy the features of the earlier layers.

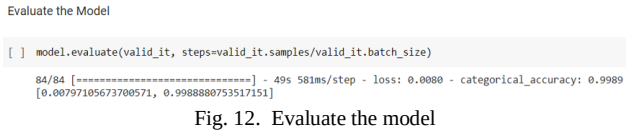

Fig. 12. Evaluate the model

The evaluate function will return a tuple, where the first value is your loss, and the second value is your accuracy. The achieved accuracy is 99.89.

5. Conclusion

A new model for classifying fruits using convolutional neural networks is proposed in this paper. The model developed has an accuracy of 99.89 % on testing data. This result is obtained after training the data with 3 fruits and 6 categories with 10904 images in the training data set and 2698 images in the testing data. This paper explores a fruits classification based on CNN

Algorithm with transfer learning approach. This paper deals with various methods and algorithms used for fruit classification. CNN better performance to attain better fruit classification with the use of softmax activation function in output layer and adam optimizer.

Fig. 12.

Fig. 13.

6. Future Scope

Hopefully in future this process can spread to more varieties of dataset of fruits as well as vegetables. Also plan to implement some other CNN based models with different activation functions and approaches to compare the accuracy on the same dataset. Would also like to work on some more features for grading and classification, which can identify types of disease and/or texture structure of fruits.

References

[1] Khatun, Mehenag & Nine, Julker & Ali, Md. Forhad & Sarker, Pritom & Turzo, Nakib. (2020). Fruits Classification using Convolutional Neural Network. 5. 1-6.

[2] Jan Steinbrener, Konstantin Posch, Raimund Leitner, Hyperspectral fruit and vegetable classification using convolutional neural networks, Computers and Electronics in Agriculture, Volume 162, pp. 364-372, 2019.

[3] X. Liu, D. Zhao, W. Jia, W. Ji, C. Ruan and Y. Sun, "Cucumber Fruits Detection in Greenhouses Based on Instance Segmentation," in IEEE Access, vol. 7, pp. 139635-139642, 2019.

[4] Guanjun Bao, Shibo Cai, Liyong Qi, Yi Xun, Libin Zhang, Qinghua Yang, Multi-template matching algorithm for cucumber recognition in natural environment, Computers and Electronics in Agriculture, Volume 127, pp. 754-762, 2016.

[5] https://www.igi-global.com/article/fruit-image-classification-using-convolutional-neural-networks/236206

[6] Fruit Veg CNN: Power- and Memory-Efficient Classification of Fruits & Vegetables Using CNN in Mobile MPSoC (techrxiv.org)

[7] Sebastian Kwiatkowski, The Fruits of Deep Learning: How Convolutional Neural Networks Support Robotic Harvesting and Yield Mapping, Towards Data Science.

[8] Shiv Ram Dubey and Anand Singh Jalal "Application of Image Processing in Fruit and Vegetable Analysis" A Review.

[9] Kavdır, I., Guyer, D. E.: Comparison of Artificial Neural Networks and Statistical Classifiers in Apple Sorting using Textural Feature, Biosystems Engg. 89,331-344, 2004.

[10] Ms. Snehal Mahajan, S. T. Patil "Optimization and Classification of Fruit using Machine Learning Algorithm", International Journal for Innovative Research in Science & Technology Vol.3- No 01 June 2016.

[11] Dubey, Shiv Ram, and A. S. Jalal. "Robust approach for fruit and vegetable classification." Procedia Engineering 38 (2012): 3449-3453.

[12] V. Leemans and M. F. Destain, "A real-time grading method of apples based on features extracted from defects," J. Food Eng., vol. 61, no. 1, pp. 83-89, Jan. 2004.

[13] T. N. Do and J. D. Fekete, "Large scale classification with support vector machine algorithms," in Proc. 6th Int. Conf. Mach. Learn. Appl., Cincinnati, OH.