Fruit Image Recognition

Sushma Sarvani Amaravadi

ITCS-5156 Spring 2023 - Devansh Desai

May 4, 2023

Abstract

This report is presented as a survey or an extension to previous work [1]. Any assertions made within are subjective and do not represent those of the original author.

Fruit image recognition is a challenging task due to the complex and diverse nature of fruits. In this study, we propose a novel approach for improving the accuracy of fruit image recognition using data augmentation and EfficientNet. Data augmentation techniques are used to increase the size of the training dataset, which helps to improve the model's ability to generalize unseen data. Our study highlights the importance of data augmentation and efficient architectures for improving the accuracy of fruit image recognition. We reviewed 4 different research papers and came up with this experiment.

1. Introduction

The paper [1] proposes a new dataset of popular fruits named Fruits-360 that can be used for deep learning, object recognition, and computer vision. The importance of having a high-quality dataset is to obtain a good classifier as most of the datasets contain object with a noisy background and this could lead to incorrect classifications.

Convolution Neural Network architecture is used to develop this project and is a commonly used and effective model for image classification tasks.

However, there are several potential limitations to this model:

- Overfitting: Since the model has many parameters, it may be prone to overfitting on the training data, which can result in poor generalization to new, unseen data.

- Transformers: Transformers are a type of neural network architecture that has been successful in natural language processing tasks but have also shown promise for computer vision tasks. (Mureșan & Oltean, 2018) They operate on a sequence of patches rather than individual pixels and use self-attention mechanisms to capture global spatial relationships.

- Limited spatial information: While CNN is designed to consider the spatial structure of the input images, it may still struggle to capture fine-grained details and spatial relationships that are important for certain types of image recognition tasks.

- Training time and computational requirements: The large number of parameters in the CNN can make training time-consuming and computationally intensive.

1.1 Problem & Challenges

The project [1] used different kinds of images of fruits in different color formats to produce identification.

While the dataset itself may be useful, there are a few potential problems and challenges that users of the dataset may encounter:

- Limited number of classes: The dataset contains images of only 60 different fruit types, which may not be sufficient for some applications.

- Limited number of samples: There are only a few hundred images for each fruit class, which may not be enough to train accurate models, especially when compared to larger datasets such as ImageNet. (Puttemans et al., 2016)

- Class imbalance: The number of images for each fruit class is not balanced, with some classes having significantly more images than others. This can lead to biased models and poor generalization to new data.

- Data quality: The images in the dataset are not uniformly high quality, with some images being poorly lit or blurry. This can make it difficult for models to accurately classify fruits.

- Limited diversity: The images in the dataset are all fruits on a white background, which may not be representative of real-world scenarios. This can lead to poor performance when models are applied to real-world images.

- Lack of annotations: The dataset does not contain annotations such as bounding boxes or segmentation masks, which can make it difficult to train models for certain tasks such as object detection or segmentation.

Motivation:

There are several motivations for improving this model:

- Better accuracy: By improving the quality of the dataset, such as by adding more images or improving the image quality, it may be possible to train more accurate models for fruit classification and other related tasks.

- Improved generalization: A more diverse and representative dataset may result in models that are better able to generalize to new data, making them more useful in real-world applications. (Selvaraj et al., 2010)

- More applications: With a better dataset, it may be possible to train models for a wider range of applications, such as object detection, segmentation, and tracking.

- Research advancements: Improving the dataset can lead to research advancements in the field of computer vision and machine learning, as it can be used as a benchmark for developing and evaluating new models.

- Economic benefits: Accurate and reliable fruit classification models can have economic benefits, such as improving the efficiency of fruit sorting and grading in the agriculture industry.

These are the results [fig 1] of using traditional CNN where we can see the percentage in the training set is greater than the test set. I am trying to utilize Data augmentation with EfficientNet CNN to see if we can improve the accuracy of the test set.

Scenario	Accuracy on training set	Accuracy on test set
Grayscale	100%	95.25%
RGB	100%	98.66%
HSV	99.99%	96.09%
HSV + Grayscale	99.99%	96.68%
HSV + Grayscale + hue/saturation change + flips	99.98%	96.44%

Fig 1 Results for accuracy

2.Background

2.1 Existing related approaches

There are several existing approaches that have been proposed to improve fruit image classification models, some of which are listed below: (Song et al., 2014)

- Transfer learning: Transfer learning is a popular approach in which a pre-trained model on a large dataset, such as ImageNet, is fine-tuned on the Fruit-Images-Dataset.

- Ensemble methods: Ensemble methods involve combining multiple models to improve the overall performance. For example, several models can be trained on different subsets of the dataset and their predictions can be combined to obtain a final prediction.

- Attention mechanisms: Attention mechanisms have been proposed to improve the model's ability to focus on relevant features in the input image. This can be especially useful when dealing with complex images, such as those with occlusions or multiple objects.

- Multi-task learning: Multi-task learning involves training a single model to perform multiple related tasks, such as fruit classification and segmentation. This can improve the model's ability to learn more meaningful representations and improve its overall performance.

2.2 Data augmentation:

Data augmentation involves generating new training images by applying various transformations to the original images, such as rotation, scaling, and flipping. This can increase the diversity of the training data and improve the model's ability to generalize to new images.

To improve upon these limitations, there are several alternative models that can be used, such as:

- ResNet: Residual Networks are a type of deep neural network that use shortcut connections to enable training of very deep architectures.

- Inception: Inception is a family of neural network architectures that use multiple convolutional filters at each layer to capture a variety of spatial scales and aspect ratios in the input image.

- EfficientNet: EfficientNet is a family of neural networks that use a combination of scaling and compound scaling techniques to achieve state-of-the-art accuracy on a wide range of image recognition tasks, while minimizing computational requirements.

3 Methods

We are performing data augmentation on the training dataset which has 67692 different fruit images. Below is the code in Figure 2 taking the training dataset and applying data augmentation.

a) Provided different parameters which helps in changing the properties of each image.

b) Rotating the image

c) Shifting the image

d) Shearing the image

e) Zooming the image

f) Flipping the image

g) Applying the brightness

h) Total number of images that needs to be augmented based on the original image

In [ ]:
```
# Define the data augmentation parameters
rotation_range = (-1, 1)
width_shift_range = (-0.01, 0.01)
height_shift_range = (-0.01, 0.01)
shear_range = (-10, 10)
zoom_range = (0.5, 0.7)
horizontal_flip = False
vertical_flip = False
```

In [ ]:
```
# Define the functions for each data augmentation technique
def rotate_image(img, rotation_range):
    angle = random.uniform(rotation_range[0], rotation_range[1])
    return img.rotate(angle)
```

In [ ]:
```
def shift_image(img, width_shift_range, height_shift_range):
    width_shift = random.uniform(width_shift_range[0], width_shift_range[1]) * img.size[0]
    height_shift = random.uniform(height_shift_range[0], height_shift_range[1]) * img.size[1]
    return img.transform(img.size, Image.AFFINE, (1, 0, width_shift, 0, 1, height_shift))
```

In [ ]:
```
def shear_image(img, shear_range):
    shear = random.uniform(shear_range[0], shear_range[1])
    return img.transform(img.size, Image.AFFINE, (1, shear, 0, 0, 1, 0))
```

In [ ]:
```
def zoom_image(img, zoom_range):
    zoom = random.uniform(zoom_range[0], zoom_range[1])
    new_size = tuple(int(dim * zoom) for dim in img.size)
    return img.resize(new_size, Image.BICUBIC)
```

In [ ]:
```
def flip_image(img, horizontal_flip, vertical_flip):
    if horizontal_flip and vertical_flip:
        return img.transpose(Image.FLIP_BOTH)
    elif horizontal_flip:
        return img.transpose(Image.FLIP_LEFT_RIGHT)
    elif vertical_flip:
        return img.transpose(Image.FLIP_TOP_BOTTOM)
    else:
        return img
```

In [ ]:
```
def apply_random_brightness(img, brightness_range=(0.5, 1.5)):
    brightness = random.uniform(brightness_range[0], brightness_range[1])
    enhancer = ImageEnhance.Brightness(img)
    return enhancer.enhance(brightness)
```

In [ ]:
```
num_augmented_per_image = 3
```

Figure 2: Parameters and methods being called for data augmentation.

Using the above parameters, we created the below method which scans each image in all the subfolders in the training folder and saves it in augumented_dir folder as shown in Figure 3.

```
In [ ]: # Define a function to apply data augmentation to images in a directory
def apply_data_augmentation(original_dir, augmented_dir, num_augmented_per_image=3):
    augmented_images = []
    labels = []

    for subdir in os.listdir(original_dir):
        if not os.path.isdir(os.path.join(original_dir, subdir)):
            continue  # Skip non-directory files

        for file in os.listdir(os.path.join(original_dir, subdir)):
            if not file.endswith('.jpg'):
                continue  # Skip non-JPEG files

            img_path = os.path.join(original_dir, subdir, file)
            img = Image.open(img_path)

            # Apply the data augmentation transformations
            img = rotate_image(img, rotation_range)
            img = shift_image(img, width_shift_range, height_shift_range)
            img = shear_image(img, shear_range)
            img = zoom_image(img, zoom_range)
            img = flip_image(img, horizontal_flip, vertical_flip)

            # Save the augmented images to the augmented directory
            img_name, img_ext = os.path.splitext(file)
            for i in range(num_augmented_per_image):
                augmented_img_name = f"{img_name}_aug{i+1}"
                augmented_dir_class = os.path.join(augmented_dir, subdir)
                os.makedirs(augmented_dir_class, exist_ok=True)
                augmented_img_path = os.path.join(augmented_dir_class, f"{augmented_img_name}{img_ext}")
                augmented_img = img.copy()
                augmented_img = apply_random_brightness(augmented_img)
                augmented_img.save(augmented_img_path)
                augmented_images.append(augmented_img_path)
                labels.append(subdir)

    return augmented_images, labels


augmented_images, labels = apply_data_augmentation(original_dir, augmented_dir, num_augmented_per_image)
```

Figure 3: Methods to apply data augmentation.

The images that were augmented were aligned with the training image shape and size and then the entire training and data augmented images were merged creating 1 new dataset for training. There is a total of 67692 in the training set. After data augmentation is applied the total images in the training set were merged in a new directory, we have 269292. Method in Figure 4.

```
n [ ]: import shutil
from PIL import Image
# Define the path for the new training directory
new_train_dir = "C:\\Users\gorti\Desktop\\project\\Fruit-Images-Dataset-master\\NewTraining\\"

# Create the new training directory if it does not exist
if not os.path.exists(new_train_dir):
    os.makedirs(new_train_dir)

# Loop through the subfolders in the original directory
for subdir in os.listdir(original_dir):
    if not os.path.isdir(os.path.join(original_dir, subdir)):
        continue  # Skip non-directory files

    # Create a subfolder in the new training directory with the same name as the original directory subfolder
    subdir_path = os.path.join(new_train_dir, subdir)
    os.makedirs(subdir_path)

    # Copy the images from the original directory subfolder to the new training directory subfolder
    for file in os.listdir(os.path.join(original_dir, subdir)):
        if not file.endswith('.jpg'):
            continue  # Skip non-JPEG files
        shutil.copy(os.path.join(original_dir, subdir, file), os.path.join(subdir_path, file))

# Loop through the subfolders in the augmented directory
for subdir in os.listdir(augmented_dir):
    if not os.path.isdir(os.path.join(augmented_dir, subdir)):
        continue  # Skip non-directory files

    # If the subfolder exists in the new training directory, copy the augmented images to it
    if os.path.exists(os.path.join(new_train_dir, subdir)):
        for file in os.listdir(os.path.join(augmented_dir, subdir)):
            if not file.endswith('.jpg'):
                continue  # Skip non-JPEG files
            img = Image.open(os.path.join(augmented_dir, subdir, file))
            img = img.resize((100, 100))  # Resize the image
            img.save(os.path.join(new_train_dir, subdir, file))  # Save the resized image to the new training directory
    # If the subfolder does not exist in the new training directory, create it and copy the augmented images to it
    else:
        subdir_path = os.path.join(new_train_dir, subdir)
        os.makedirs(subdir_path)
        for file in os.listdir(os.path.join(augmented_dir, subdir)):
            if not file.endswith('.jpg'):
                continue  # Skip non-JPEG files
            img = Image.open(os.path.join(augmented_dir, subdir, file))
            img = img.resize((100, 100))  # Resize the image
            img.save(os.path.join(subdir_path, file))  # Save the resized image to the new training directory
```

Figure 4: Merging the train data images with augmented images.

Once the new training data was created the data is sent through training using below methods

train_datagen is used for data augmentation of the training dataset. It rescales the pixel values of the images to a range of 0 to 1, performs random horizontal flipping, random zooming, and shearing.

test_datagen is used for scaling the pixel values of the test dataset to a range of 0 to 1. It does not perform any data augmentation. Refer to Figure 5(a),

```
In [21]: train_datagen = ImageDataGenerator(rescale = 1./255,
                                            shear_range = 0.3,
                                            horizontal_flip=True,
                                            vertical_flip=False,
                                            zoom_range = 0.3
                                            )
         test_datagen  = ImageDataGenerator(rescale = 1./255)
```

Figure 5(a): train and test dataset creator

train_generator is generating batches of images from the new_train_dir directory, which contains the training images that have been augmented. The images are resized to (img_width, img_height), and the batch size is set to train_batch_size. The color_mode is set to "rgb" to indicate that the images are in RGB format, and class_mode is set to "categorical" to indicate that the labels for the images are in categorical format.

```
In [22]: train_generator = train_datagen.flow_from_directory(new_train_dir,
                                                             target_size=(img_width, img_height),
                                                             batch_size = train_batch_size,
                                                             color_mode= "rgb",
                                                             class_mode = "categorical")
```

Figure 5(b): train_generator

test_generator generates batches of augmented image data from the test dataset directory. It takes the path to the test dataset directory and a set of image preprocessing parameters and generates batches of images during the model evaluation phase.

```
# Create the data generator for the testing data

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=(img_width, img_height),
                                                  batch_size = test_batch_size,
                                                  color_mode= "rgb",
                                                  class_mode = "categorical")
```

Figure 5(c): test_generator

We used EfficientNet CNN algorithm to train the model and provide better results as shown below Figure 6

```
# Load the EfficientNetB0 model and add layers to it
base_model = EfficientNetB0(include_top=False, weights='imagenet', input_shape=input_shape)
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.summary()
```

Figure 6: Adding layers to the EfficientNet Model

We are saving the training model as a layer and compiling the model to provide loss and accuracy metrics using a custom loss function as shown in below Figure 7

```
# Freeze the layers in the base model
for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer=Adam(learning_rate=0.001), loss=custom_loss, metrics=['accuracy'])
from keras.callbacks import EarlyStopping
# Set up early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)
```

Figure 7: Locking layers

The function takes two input arguments, y_true and y_pred, which represent the true labels and predicted labels, respectively. From Figure 8

- The first line of the function defines a small value called "epsilon" to avoid division by zero errors.
- The second line of the function clips the predicted values to be between epsilon and 1-epsilon to avoid numerical instability.
- The third line of the function computes the cross-entropy loss between the true labels and predicted labels using the tf.reduce_sum() and tf.math.log() functions.Finally, the loss value is returned from the function.

```
from keras import backend as K
K.set_epsilon(1e-07)
def custom_loss(y_true, y_pred):
    epsilon = tf.keras.backend.epsilon()
    y_pred = tf.clip_by_value(y_pred, epsilon, 1.0 - epsilon)
    loss = -tf.reduce_sum(y_true * tf.math.log(y_pred), axis=-1)
    return loss
```

Figure 8: Setting up Custom loss function

4. Experiments

For this experiment we are trying to use the actual dataset instead of augmented data as it is taking a lot of time. This code trains the model using the training data provided by train_generator and validates it using the test data provided by test_generator.

```
hist = model.fit(train_generator,
                 steps_per_epoch=1340,
                 epochs=50,
                 validation_data=test_generator,
                 validation_steps=454)
```

Below are the tables of different datasets being used to train and measure accuracy of the model.

Training data set Coverage for Augmented data: total samples 269292

Subfolder Name	Number of JPEG Images
Apple Braeburn	492
Apple Crimson Snow	1776
Apple Golden 1	1920
Apple Golden 2	1968
Apple Golden 3	1924
Apple Granny Smith	1968
Apple Pink Lady	1824
Apple Red 1	1968
Apple Red 2	1968
Apple Red 3	1716
Apple Red Delicious	1960
Apple Red Yellow 1	1968
Apple Red Yellow 2	2688
Apricot	1968
Avocado	1708
Avocado ripe	1964
Banana	1960
Banana Lady Finger	1800
Banana Red	1960
Beetroot	1800
Blueberry	1848
Cactus fruit	1960
Cantaloupe 1	1968
Cantaloupe 2	1968
Carambula	1960
Cauliflower	2808
Cherry 1	1968
Cherry 2	2952
Cherry Rainier	2952
Cherry Wax Black	1968
Cherry Wax Red	1968
Cherry Wax Yellow	1968
Chestnut	1800
Clementine	1960
Cocos	1960
Corn	1800

Corn Husk	1848
Cucumber Ripe	1568
Cucumber Ripe 2	1872
Dates	1960
Eggplant	1872
Fig	2808
Ginger Root	1188
Granadilla	1960
Grape Blue	3936
Grape Pink	1968
Grape White	1960
Grape White 2	1960
Grape White 3	1968
Grape White 4	1884
Grapefruit Pink	1960
Grapefruit White	1968
Guava	1960
Hazelnut	1856
Huckleberry	1960
Kaki	1960
Kiwi	1864
Kohlrabi	1884
Kumquats	1960
Lemon	1968
Lemon Meyer	1960
Limes	1960
Lychee	1960
Mandarine	1960
Mango	1960
Mango Red	1704
Mangostan	1200
Maracuja	1960
Melon Piel de Sapo	2952
Mulberry	1968
Nectarine	1968
Nectarine Flat	1920
Nut Forest	2616
Nut Pecan	2136
Onion Red	1800
Onion Red Peeled	1780
Onion White	1752
Orange	1916
Papaya	1968
Passion Fruit	1960
Peach	1968
Peach 2	2952
Peach Flat	1968
Pear	1968
Pear 2	2784
Pear Abate	1960
Pear Forelle	2808
Pear Kaiser	1200
Pear Monster	1960
Pear Red	2664
Pear Stone	2844
Pear Williams	1960
Pepino	1960
Pepper Green	1776
Pepper Orange	2808
Pepper Red	2664
Pepper Yellow	2664
Physalis	1968
Physalis with Husk	1968
Pineapple	1960
Pineapple Mini	1972
Pitahaya Red	1960
Plum	1788
Plum 2	1680
Plum 3	3600
Pomegranate	1968
Pomelo Sweetie	1800
Potato Red	1800
Potato Red Washed	1812
Potato Sweet	1800
Potato White	1800
Quince	1960
Rambutan	1968

Raspberry	1960
Redcurrant	1968
Salak	1960
Strawberry	1968
Strawberry Wedge	2952
Tamarillo	1960
Tangelo	1960
Tomato 1	2952
Tomato 2	2688
Tomato 3	2952
Tomato 4	1916
Tomato Cherry Red	1968
Tomato Heart	2736
Tomato Maroon	1468
Tomato not Ripened	1896
Tomato Yellow	1836
Walnut	2940
Watermelon	1900
Total	**269292**

Training data set Coverage: total samples: 67692

Fruit Name	Sample Size of train data
Apple Braeburn	492
Apple Crimson Snow	444
Apple Golden 1	480
Apple Golden 2	492
Apple Golden 3	481
Apple Granny Smith	492
Apple Pink Lady	456
Apple Red 1	492
Apple Red 2	492
Apple Red 3	429
Apple Red Delicious	490
Apple Red Yellow 1	492
Apple Red Yellow 2	672
Apricot	492
Avocado	427
Avocado ripe	491
Banana	490
Banana Lady Finger	450
Banana Red	490
Beetroot	450
Blueberry	462
Cactus fruit	490
Cantaloupe 1	492
Cantaloupe 2	492
Carambula	490
Cauliflower	702
Cherry 1	492
Cherry 2	738
Cherry Rainier	738
Cherry Wax Black	492
Cherry Wax Red	492
Cherry Wax Yellow	492
Chestnut	450
Clementine	490
Cocos	490
Corn	450
Corn Husk	462
Cucumber Ripe	392
Cucumber Ripe 2	468
Dates	490
Eggplant	468
Fig	702
Ginger Root	297
Granadilla	490
Grape Blue	984
Grape Pink	492
Grape White	490
Grape White 2	490
Grape White 3	492
Grape White 4	471
Grapefruit Pink	490
Grapefruit White	492
Guava	490
Hazelnut	464

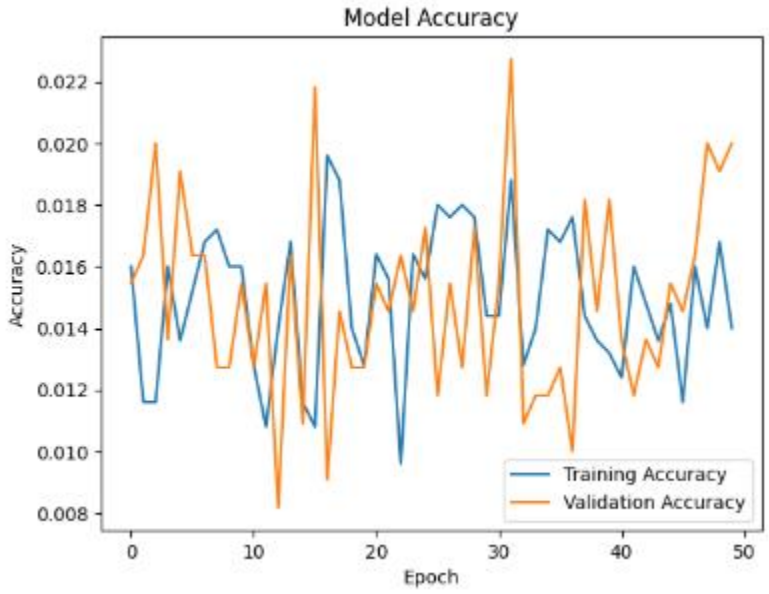
Huckleberry	490
Kaki	490
Kiwi	466
Kohlrabi	471
Kumquats	490
Lemon	492
Lemon Meyer	490
Limes	490
Lychee	490
Mandarine	490
Mango	490
Mango Red	426
Mangostan	300
Maracuja	490
Melon Piel de Sapo	738
Mulberry	492
Nectarine	492
Nectarine Flat	480
Nut Forest	654
Nut Pecan	534
Onion Red	450
Onion Red Peeled	445
Onion White	438
Orange	479
Papaya	492
Passion Fruit	490
Peach	492
Peach 2	738
Peach Flat	492
Pear	492
Pear 2	696
Pear Abate	490
Pear Forelle	702
Pear Kaiser	300
Pear Monster	490
Pear Red	666
Pear Stone	711
Pear Williams	490
Pepino	490
Pepper Green	444
Pepper Orange	702
Pepper Red	666
Pepper Yellow	666
Physalis	492
Physalis with Husk	492
Pineapple	490
Pineapple Mini	493
Pitahaya Red	490
Plum	447
Plum 2	420
Plum 3	900
Pomegranate	492
Pomelo Sweetie	450
Potato Red	450
Potato Red Washed	453
Potato Sweet	450
Potato White	450
Quince	490
Rambutan	492
Raspberry	490
Redcurrant	492
Salak	490
Strawberry	492
Strawberry Wedge	738
Tamarillo	490
Tangelo	490
Tomato 1	738
Tomato 2	672
Tomato 3	738
Tomato 4	479
Tomato Cherry Red	492
Tomato Heart	684
Tomato Maroon	367
Tomato not Ripened	474
Tomato Yellow	459
Walnut	735
Watermelon	475
Total	67692

Testing dataset coverage: total samples: 22688

Fruit Name	Sample Size of Test Data
Apple Braeburn	164
Apple Crimson Snow	148
Apple Golden 1	160
Apple Golden 2	164
Apple Golden 3	161
Apple Granny Smith	164
Apple Pink Lady	152
Apple Red 1	164
Apple Red 2	164
Apple Red 3	144
Apple Red Delicious	166
Apple Red Yellow 1	164
Apple Red Yellow 2	219
Apricot	164
Avocado	143
Avocado ripe	166
Banana	166
Banana Lady Finger	152
Banana Red	166
Beetroot	150
Blueberry	154
Cactus fruit	166
Cantaloupe 1	164
Cantaloupe 2	164
Carambula	166
Cauliflower	234
Cherry 1	164
Cherry 2	246
Cherry Rainier	246
Cherry Wax Black	164
Cherry Wax Red	164
Cherry Wax Yellow	164
Chestnut	153
Clementine	166
Cocos	166
Corn	150
Corn Husk	154
Cucumber Ripe	130
Cucumber Ripe 2	156
Dates	166
Eggplant	156
Fig	234
Ginger Root	99
Granadilla	166
Grape Blue	328
Grape Pink	164
Grape White	166
Grape White 2	166
Grape White 3	164
Grape White 4	158
Grapefruit Pink	166
Grapefruit White	164
Guava	166
Hazelnut	157
Huckleberry	166
Kaki	166
Kiwi	156
Kohlrabi	157
Kumquats	166
Lemon	164
Lemon Meyer	166
Limes	166
Lychee	166
Mandarine	166
Mango	166
Mango Red	142
Mangostan	102
Maracuja	166
Melon Piel de Sapo	246
Mulberry	164
Nectarine	164
Nectarine Flat	160
Nut Forest	218
Nut Pecan	178
Onion Red	150
Onion Red Peeled	155
Onion White	146

Orange	160	Pomegranate	164
Papaya	164	Pomelo Sweetie	153
Passion Fruit	166	Potato Red	150
Peach	164	Potato Red Washed	151
Peach 2	246	Potato Sweet	150
Peach Flat	164	Potato White	150
Pear	164	Quince	166
Pear 2	232	Rambutan	164
Pear Abate	166	Raspberry	166
Pear Forelle	234	Redcurrant	164
Pear Kaiser	102	Salak	162
Pear Monster	166	Strawberry	164
Pear Red	222	Strawberry Wedge	246
Pear Stone	237	Tamarillo	166
Pear Williams	166	Tangelo	166
Pepino	166	Tomato 1	246
Pepper Green	148	Tomato 2	225
Pepper Orange	234	Tomato 3	246
Pepper Red	222	Tomato 4	160
Pepper Yellow	222	Tomato Cherry Red	164
Physalis	164	Tomato Heart	228
Physalis with Husk	164	Tomato Maroon	127
Pineapple	166	Tomato not Ripened	158
Pineapple Mini	163	Tomato Yellow	153
Pitahaya Red	166	Walnut	249
Plum	151	Watermelon	157
Plum 2	142	Total	22688
Plum 3	304		

We were able to obtain better accuracy for validation based on he training dataset as shown below



Training the model with 269292 images is taking a vast amount of time and is still in progress after 72 hours. Therefore, we must create these results based on the training dataset of 67692. Using EfficientNet CNN we can see the validation accuracy is better for the model.

Conclusion:

Based on the above experiment more training data and efficient CNN algorithms the accuracy increases providing better results.

References:

Mureșan, H., & Oltean, M. (2018). Fruit recognition from images using deep learning. *Acta Universitatis Sapientiae, Informatica*, *10*, 26–42. https://doi.org/10.2478/ausi-2018-0002

Puttemans, S., Vanbrabant, Y., Tits, L., & Goedemé, T. (2016, December 15). *Automated visual fruit detection for harvest estimation and robotic harvesting*. https://doi.org/10.1109/IPTA.2016.7820996

Selvaraj, A., Shebiah, N., Nidhyananthan, S., & Ganesan, L. (2010). Fruit Recognition using Color and Texture Features. *Journal of Emerging Trends in Computing and Information Sciences*, *1*, 90–94.

Song, Y., Glasbey, C. A., Horgan, G. W., Polder, G., Dieleman, J. A., & van der Heijden, G. W. A. M. (2014). Automatic fruit recognition and counting from multiple images. *Biosystems Engineering*, *118*, 203–215. https://doi.org/10.1016/j.biosystemseng.2013.12.008