University of Sri Jayewardenepura

Machine Learning

Semester 01 - Year 03 -2020

Fruit Classification from Images

Machine Learning

Assignment 01

Submitted by:

M.G.T.M. Kumara (Leader)	AS2017418
J.M.D.N.L. Jayamaha	AS2017391
J.D.K.N. Jayakodi	AS2017389
A.H.R. Lankapriya	AS2017423
T.D.K. Buddika	AS2017332

(Date of submission / March 31st 2021)

Table of Contents

Abstract.
In this paper We present the results of some numerical experiment for training a neural network to detect fruits. We discuss the reason why we chose to use fruits in this project by proposing a few applications that could use such classifier.

Keywords: Neutral network, Object recognition, Computer vision, fruits dataset, image processing

Introduction

Problem Statement

To build a robust system to recognize the fruits according to the color of fruits, intensity of fruits, shape of fruits and texture of the fruits. Sri Lanka is one of the most renown sort of tourist destinations in the world. The diversity of temperature, vegetation and scenery grab a large sense towards Sri Lanka and becomes one of the demanding countries among nature lovers and fans of wildlife because tourism offers safaris, nature walks and camping and that would be a marvelous experience for them. While enjoying, they prefer to taste local food specially fruits as here in Sri Lanka, we have difference varieties and types of fruits which are tasty and have medicinal value. But we never really give them the fact that how our island is home for an immense variety of fruits until many of the tourists asking about what fruits we have to try them while they are spending their vacations. Most get surprised and made curiosity after seeing those various fruits in the Sri Lankan markets which are very rare and some names are not even heard before. So, we are going to promote local fruits among tourists using image processing technique to give better understanding about fruits and its value just using an image of a fruit.

Significance of the research

Sri Lanka is famous for its various types of fruits, because different areas of the country have its own weather conditions. Foreigners and also the local travelers basically buy fruits while there are going on a journey. But there are many types of them and specially foreigners can’t remember those names and significance value of every fruit. So, if they have an method which gives them the information of fruits which they can buy in the local market, it will help them to buy the fruits which they want the most and will help to deal with local market sellers who don't know the foreign languages. As a part of research area, we develop this fruits classification using convolutional neural network and deep learning technique which is a class of machine learning.

The difficult task was finding images image data; in this case the dataset was named Fruits-360 more helpful and It can be downloaded from the addresses pointed by references [20] and [21]. Currently the set contains 90380 images of 131 fruits and vegetables and it is constantly updated with images of new fruits and vegetables as soon as the authors have accesses to them. The reader is inspired to access the newest version of the dataset from the above indicated addresses. Having a high-quality dataset is important for obtaining an honest classifier. This contains the object and they were removed noisy background. This could cause cases where changing the background will cause the wrong classification of the thing. As a second objective we've trained a neural network that's capable of identifying fruits from

images. This is a part of a more complex project that has the target of obtaining a classifier which will identify a way wider array of objects from images.

First step in creating such application is to properly identify the objects. The software has been released later in 2017 as a feature of Google Assistant and Google Photos apps. Currently the identification of objects is predicated on a deep neural network. Such a network would have numerous applications across multiple domains like autonomous navigation, modeling objects, controlling processes or human-robot interactions. The area we are most curious about is creating an autonomous robot which will perform more complex tasks than a daily industrial robot. An example of this is often a robot which will perform inspections on the aisles of stores so as to spot out of place items or under-stocked shelves. Furthermore, this robot might be enhanced to be ready to interact with the products in order that it can solve the issues on its own. Another area during which this research can provide benefits is autonomous fruit harvesting. While there are several papers on this subject already, from the simplest of our knowledge, they specialize in few species of fruits or vegetables. In this paper we plan to create a network which will classify a spread of species of fruit, thus making it useful in more scenarios. As the start of this project, we chose the task of identifying fruits for several reasons. On one side, fruits have certain categories that are hard to differentiate, just like the citrus genus, that contains oranges and grapefruits. Thus, we would like to ascertain how well can a man-made intelligence complete the task of classifying them. Another reason is that fruits are fairly often found in stores, in order that they function an honest start line for the previously mentioned a couple of outstanding achievements obtained using deep learning for fruits recognition, followed by a presentation of the concept of deep learning. In the second part we describe the Fruits-360 dataset: how it had been created and what it contains. In the third part we'll present the framework utilized in this project - TensorFlow and therefore the reasons we chose it. Following the framework presentation, we'll detail the structure of the neural network that we used. We also describe the training and testing data used also because the obtained performance. Finally, we'll conclude with a couple of plans on the way to improve the results of this project. Source code is listed in the Appendix.

Related work

In this section we review several previous attempts to use neural networks and deep learning for fruits recognition. A method for recognizing and counting fruits from images in cluttered greenhouses is presented in [28]. The targeted plants are peppers with fruits of complex shapes and ranging colors almost like the plant canopy. The aim of the appliance is to locate and count green and red pepper fruits on large, dense pepper plants growing during a greenhouse. The training and validation data utilized during this paper consists of 28000 images of over 1000 plants and their fruits. The used method to locate and count the peppers is

two-step: within the initiative, the fruits are located during a single image and during a second step multiple views are combined to extend the detection rate of the fruits. The approach to seek out the pepper fruits during a single image is predicated on a mixture of (1) finding points of interest, (2) applying a posh high-dimensional feature descriptor of a patch round the point of interest and (3) employing a so-called bag-of-words for classifying the patch. Paper presents a completely unique approach for detecting fruits from images using deep neural networks. For this purpose, the authors adapt a Faster Region-based convolutional network. The objective is to make a neural network that might be employed by autonomous robots which will harvest fruits.

The network is trained using RGB and NIR (near infrared) images. The combination of the RGB and NIR models is completed in 2 separate cases: early and late fusion. Early fusion implies that the input layer has 4 channels: 3 for the RGB image and one for the NIR image. Late fusion uses 2 independently trained

models that are merged by obtaining predictions from both models and averaging the results. The result is a multi-modal network. which obtains far better performance than the prevailing networks. On the topic of autonomous robots used for harvesting, paper [1] shows a network trained to recognize fruits in an orchard. This is a very difficult task because so as to optimize operations, images that span many fruit trees must be used. In such images, the quantity of fruits is often large, within the case of almonds up to 1500 fruits per image. Also, because the pictures are taken outside, there's tons of variance in luminosity, fruit size, clustering and consider point. Like in paper [25], this project makes use of the Faster Region-based convolutional network, which is presented during a detailed view in paper [24]. Related to the automated harvest of fruits, article [22] presents a way of detecting ripe strawberries and apples from orchards. The paper also highlights existing methods and their performance. In [11] the authors compile an inventory of the available state of the art methods for harvesting with the help of robots. They also analyze the method and propose ways to improve them. In [2] one can see a way of generating synthetic images that are highly almost like empirical images. Specifically, this paper introduces a way for the generation of large-scale semantic segmentation datasets on a plant-part level of realistic agriculture scenes, including automated per-pixel class and depth labeling. One purpose of such synthetic dataset would be too bootstrap or pre-train computer vision models, which are finetuned thereafter on a smaller empirical image dataset. Similarly, in paper [23] we will see a network trained on synthetic images which will count the number of fruits in images without actually detecting where they're within the image. Another paper, [4], uses two back propagation neural networks trained on images with apple" Gala" variety trees so on predict the yield for the upcoming season. For this task, four features are extracted from images: total cross-sectional area of fruits, fruit number, total cross-section area of small fruits, and cross-sectional area of foliage. Paper [10] presents an analysis of fruit detectability in regard to the angle of the camera when the image was taken. Based on this research, it had been concluded that the fruit detectability was the very best on front views and searching with a zenith angle of 60_ upwards. In papers [27, 37, 15] we can see an approach to detecting fruits based on color, shape and texture. They highlight the problem of correctly classifying similar fruits of various species. They propose combining existing methods using the feel, shape and color of fruits to detect regions of interest from images. Similarly, in [18] a way combining shape, size and color, texture of the fruits alongside a k nearest neighbor algorithm is employed to extend the accuracy of recognition.

Deep learning

In the area of image recognition and classification, the foremost successful results were obtained using artificial neural networks [6, 30]. These networks form the idea for many deep learning models. Deep learning may be a class of machine learning algorithms that use multiple layers that contain nonlinear processing units [26]. Each level learns to transform its input data into a slightly more abstract and composite representation [6]. Deep neural networks have managed to outperform other machine learning algorithms. They also achieved the primary superhuman pattern recognition in certain domains [5]. This is further reinforced by the very fact that deep learning is taken into account as a crucial step towards obtaining Strong AI. Secondly, deep neural networks - specifically convolutional neural networks - are proved to get great leads to the sector of image recognition. In the rest of this section, we will briefly describe some models of deep artificial neural networks alongside some results for a few related problems.

Convolutional neural networks

Convolutional neural networks (CNN) are a neighborhood of the deep learning models. Such a network is often composed of convolutional layers, pooling layers, ReLU layers, fully connected layers and loss layers [34]. In a typical CNN architecture, each convolutional layer is followed by a Rectified linear measure (ReLU) layer, then a Pooling layer then one or more convolutional layer and eventually, one or more fully connected layer. A characteristic that sets apart the CNN from a daily neural network is taking under consideration the structure of the pictures while processing them. Note that a daily neural network converts the input during a one-dimensional array which makes the trained classifier less sensitive to positional changes. Among the simplest results obtained on the MNIST [13] dataset is completed by using multi-column deep neural networks. As described in paper [7], they use multiple maps per layer with many layers of non-linear neurons. Even if the complexity of such networks makes them harder to coach, by using graphical processors and special code written for them. The structure of the network uses winner-take-all neurons with max pooling that determine the winner neurons. Another paper [16] further reinforces the thought that convolutional networks have obtained better accuracy within the domain of computer vision. In paper [29] an all-convolutional network that gains excellent performance on CIFAR-10 [12] is described intimately. The paper proposes the replacement of pooling and fully connected layers with equivalent convolutional ones. This may increase the number of parameters and adds inter-feature dependencies however it is often mitigated by using smaller convolutional layers within the network and acts as a sort of regularization. In what follows we'll describe each of the layers of a CNN network.

Convolutional layers

Convolutional layers are named after the convolution operation. In mathematics convolution is an operation on two functions that produces a 3rd function that's the modified (convoluted) version of 1 of the first functions. The resulting function gives in integral of the pointwise multiplication of the 2 functions as a function of the quantity that one among the first functions is translated [33].

A convolutional layer consists of groups of neurons that structure kernels. The kernels have alittle size but they always have an equivalent depth because the input. The neurons from a kernel are connected to alittle region of the input, called the receptive field, because it's highly incident to link all neurons to all or any previous outputs within the case of inputs of high dimensions like images. For example, a 100 x 100 image has 10000 pixels and if the primary layer has 100 neurons, it might end in 1000000 parameters. Instead of each neuron having weights for the complete dimension of the input, a neuron holds weights for the dimension of the kernel input. The kernels slide across the width and height of the input, extract high level features and produce a 2-dimensional activation map. The stride at which a kernel slides is given as a parameter. The output of a convolutional layer is formed by stacking the resulted activation maps which in turned is employed to define the input of subsequent layer. Applying a convolutional layer over a picture of size 32 X 32 leads to an activation map of size 28 X 28. If we apply more convolutional layers, the size will be further reduced, and, as a result the image size is drastically reduced which produces loss of information and the vanishing gradient problem. To correct this, we use padding. Padding increases the dimensions of an input file by filling constants around input file. In most of the cases, this constant is zero therefore the operation is known as zero padding.” Same” padding means the output feature map has an equivalent spatial dimension as the input feature map. This tries to pad evenly left and right, but if the number of columns to be added is odd, it'll add an additional column to the proper. “Valid” padding is equivalent to no padding. The strides cause a kernel to skip over pixels in an image and not include them in the output. The strides determine how a convolution operation works with a kernel when a larger image and more complex kernel are used. As a kernel is sliding the input, it's using the strides parameter to work out what percentage positions to skip. ReLU layer, or Rectified Linear Units layer, applies the activation

function max (0, x). It doesn't reduce the dimensions of the network, but it increases its nonlinear properties.

Pooling layers

Pooling layers were used on one hand to scale back the spatial dimensions of the representation and to scale back the quantity of computation wiped out the network. The other use of pooling layers is to regulate overfitting. The most used pooling layer has filters of size 2 x 2 with a stride 2. This electively reduces the input to quater of its original size.

Fully connected layers

Fully connected layers are layers from a daily neural network. Each neuron from a totally connected layer is linked to every output of the previous layer. The operations behind a convolutional layer are an equivalent as during a fully connected layer. Thus, it's possible to convert between the two.

Loss layers

Loss layers are wont to penalize the network for deviating from the expected output. This is normally the last layer of the network. Various loss function exists: SoftMax is employed for predicting a category from multiple disjunct classes, sigmoid cross-entropy is employed for predicting multiple independent probabilities (from the [0, 1] interval).

Recurrent neural network

Another deep learning algorithm is that the recursive neural network [16]. The paper proposes an improvement to the favored convolutional network within the sort of a recurrent convolutional network. In this quite architecture an equivalent set of weights is recursively applied over some data. Traditionally, recurrent networks are wont to process sequential data, handwriting or speech recognition being the foremost known examples. By using recurrent convolutional layers with some max pool layers in between them and a final global max pool layer at the highest several advantages are obtained. Firstly, within a layer, every unit takes under consideration the state of units in an increasingly larger area around it. Secondly, by having recurrent layers, the depth of the network is increased without adding more parameters. Recurrent networks have shown good leads to tongue processing.

Deep network

Yet another model that's a part of the deep learning algorithms is that the deep belief network [14]. A deep belief network may be a probabilistic model composed by multiple layers of hidden units. The usages of a deep belief network are an equivalent because the other presented networks but also can be wont to pre-train a deep neural network so as to improve the initial values of the weights. This process is vital because it can improve the standard of the network and may reduce training times. Deep belief networks can be combined with convolutional ones in order to obtain convolutional deep belief networks which exploit the advantages ordered by both types of architectures.

Fruits-360 data set

In this section we describe how the info set was created and what it contains. The images were obtained by filming the fruits while they're rotated by a motor then extracting frames. Fruits were planted within the shaft of a low-speed motor (3 rpm) and a brief movie of 20 seconds was recorded. Behind the fruits we placed a white sheet of paper as background. However, thanks to the variations within the lighting conditions, the background wasn't uniform and that we wrote a fanatical algorithm which extract the fruit from the background. This algorithm is of flood fill type: we start from each fringe of the image and that we mark all pixels there, then we mark all pixels found within the neighborhood of the already marked pixels that

the distance between colors is a smaller amount than a prescribed value. we repeat the previous step until no more pixels are often marked. All marked pixels are considered as being background (which is then crammed with white) and therefore the remainder of pixels are considered as belonging to the thing. The maximum value for the space between 2 neighbor pixels may be a parameter of the algorithm and is about (by trial and error) for every movie. Fruits were scaled to suit a 100x100 pixels image. Other datasets (like MNIST) use 28x28 images, but we feel that tiny size is detrimental once you have too similar objects (a red cherry looks very almost like a red apple in small images). Our future plan is to figure with even larger images, but this may require far more longer training times. To understand the complexity of background-removal process we've depicted in Figure 1 a fruit with its original background and after the background was removed and therefore the fruit was scaled right down to 100 x 100 pixels.

The resulted dataset has 82110 images of fruits and vegetables spread across 120 labels. Each image contains a single fruit or vegetable. Separately, the dataset contains another 103 images of multiple fruits. The data set is available on GitHub [20] and Kaggle [21]. The labels and therefore the number of images for training are given in Table 1. Table 1: Number of images for each fruit. There are multiple sorts of apples each of them being considered as a separate object. We did not find the scientific/popular name for each apple so we labeled with digits (e.g., apple red 1, apple red 2 etc.).

Fruit	Training Images
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	492
Apple Red Delicious	429
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
Cron	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	492
Strawberry	738
Strawberry Wedge	490
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
Veralu	86
Walnut	735
Watermelon	475

TensorFlow library

For the aim of implementing, training and testing the network described during this paper we used the TensorFlow library [32]. This is an opensource framework for machine learning created by Google for numerical computation using data flow graphs. Nodes within the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays called tensors.

The main components during a TensorFlow system are the client, which uses the Session interface to speak with the master, and one or more worker processes, with each worker process responsible for arbitrating access to a minimum of one or more computational devices (such as CPU cores or GPU cards) and for executing graph nodes on those devices as instructed by the master.

TensorFlow owns some powerful features such as: it allows computation mapping to multiple machines, unlike most other similar frameworks; it's inbuilt support for automatic gradient computation; it can partially execute subgraphs of the whole graph and it can add constraints to devices, like placing nodes on devices of a particular type, make sure that two or more objects are placed in the same space etc.

TensorFlow is employed in several projects, like the Inception Image Classification Model [31]. This project introduced a state-of-the-art network for classification and detection within the ImageNet Large-Scale Visual Recognition Challenge 2014. In this project the usage of the computing resources is improved by adjusting the network width and depth while keeping the computational budget constant [31].

Another project that employs the TensorFlow framework is DeepSpeech, developed by Mozilla. It is an open-source Speech-To-Text engine based on Baidu’s Deep Speech architecture [9]. The architecture may

be a state-of-the-art recognition system developed using end-to-end deep learning. It is simpler that other architectures and doesn't need hand designed components for ground noise, reverberation or speaker variation.
We will present the foremost important utilized methods and data types from TensorFlow alongside a brief description for every of them.
A convolutional layer is defined like this:

The structure of the neural network used in experiments

For this project we used a convolutional neural network. As previously described this sort of network makes use of convolutional layers, pooling layers, ReLU layers, fully connected layers and loss layers. In a typical CNN architecture, each convolutional layer is followed by a Rectified linear measure (ReLU) layer, then a Pooling layer then one or more convolutional layer and eventually, one or more fully connected layer. Note again that a characteristic that sets apart the CNN from a daily neural network is taking under consideration the structure of the pictures while processing them. A regular neural network converts the input during a one-dimensional array which makes the trained classifier less sensitive to positional changes. The input that we used consists of ordinary RGB images of size 100 x 100 pixels.

The neural network that we used in this project has the structure given below

- The first layer (Convolution #1) is a convolutional layer which applies 16 5 x 5 filters. On this layer we apply max pooling with a filter of
- shape 2 x 2 with stride 2 which specifies that the pooled regions do not overlap (Max-Pool #1). This also reduces the width and height to 50 pixels each.
- The second convolutional (Convolution #2) layer applies 32 5 x 5 filters which outputs 32 activation maps. We apply on this layer the same kind of max pooling (Max-Pool #2) as on the first layer, shape 2 x 2 and stride 2.
- The third convolutional (Convolution #3) layer applies 64 5 x 5 filters. Following is another max pool layer (Max-Pool #3) of shape 2 x 2 and stride 2.
- The fourth convolutional (Convolution #4) layer applies 128 5 x 5 filters after which we apply a final max pool layer (Max-Pool #4).
- Because of the four max pooling layers, the dimensions of the representation have each been reduced by a factor of 16, therefore the fifth layer, which is a fully connected layer (Fully Connected #1), has 7 x 7 x 16 inputs.
- This layer feeds into another fully connected layer (Fully Connected #2) with 1024 inputs and 256 outputs.
- The last layer is a softmax loss layer (Softmax) with 256 inputs. The number of outputs is equal to the number of classes. (132)

Evaluation Measures

After training the model, we will apply the evaluation measures to check that how the model is getting predictions. We will use the following evaluation measures to evaluate the performance of the model:

- Accuracy

- Plots of training and validation scores

Accuracy graph

any layers as well as the introduction of new layers can provide completely different results. Another option is to replace all layers with convolutional layers. This has been shown to provide some improvement over the networks that have fully connected layers in their structure. A consequence of replacing all layers with convolutional ones is that there will be an increase in the number of parameters for the network [29]. Another possibility is to replace the rectified linear units with exponential linear units. According to paper [8], this reduces computational complexity and add significantly better generalization performance than rectified linear units on networks with more than 5 layers. We would like to try out these practices and also to try to find new configurations that provide interesting results.
In the near future we plan to create a mobile application which takes pictures of fruits and labels them accordingly. Another objective is to expand the data set to include more fruits. This is a more time-consuming process since we want to include items that were not used in most others related papers.

Technical Approach

We are using high level programming language, python language in the implementations and Jupyter Notebook that support the machine learning and data science projects. We will build TensorFlow based model. Also, mainly We will use Fruits360 dataset and also, we use some of the endemic fruits on rural areas in Sri Lanka. The dataset providers provide the training and test data separately and we have to be separate endemic fruits according to them. After training on the model, we will evaluate the model to check the performance of trained model.

Importing Libraries

Libraries are important and we call them to perform the different actions on our data and for training the models.

It's a first step to load the library to perform the specific task

We are using the following versions of the libraries:

- numpy == 1.18.5
- tensorflow ==1.7.0
- keras == 2.4.3
- matplotlib ==3.3.2

Explore data analysis

In [1]:

```
# here we have imported necessary libraries
# functions for interacting with the operating system
import os
# imorting library for fast numerical computing
import tensorflow as tf
# defining model about linear stack of layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D
from tensorflow.keras.layers import Activation, Dense, Flatten, Dropout
# define keras optimizer
from tensorflow.keras.optimizers import RMSprop
# import ptrprocesser for image data augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# import ModelCheckpoint for continously save model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img
import keras_preprocessing
from keras_preprocessing import image
from sklearn.datasets import load_files
import matplotlib.image as mpimg
import numpy as np
# magic function that renders the figure in a notebook
%matplotlib inline
# plotting a figure
import matplotlib.pyplot as plt
from keras.utils import np_utils
```

```
Using TensorFlow backend.
```

Exploration of data is not necessary for training the model but it's a good practice to look at the dataset so that we can analyze that what type of data we are using and how we can handle it.

Sample images

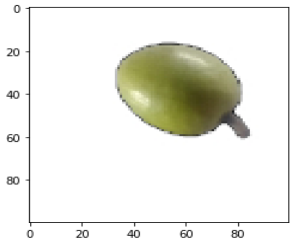
In [2]:

```
#view sample training data and shape of data
img = mpimg.imread(r'C:\Users\USER\Desktop\fruit_recognition\fruits-360\Training\Veralu\Veralu (12).jpg')
print(img.shape)
plt.imshow(img)
```

```
(100, 100, 3)
```

Out[2]:

```
<matplotlib.image.AxesImage at 0x25c70036ac8>
```



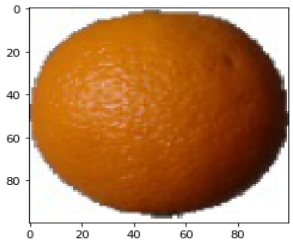
In [3]:

```
#view sample training data and shape of data
img = mpimg.imread(r'C:\Users\USER\Desktop\fruit_recognition\fruits-360\Training\Orange\0_100.jpg')
print(img.shape)
plt.imshow(img)
```

```
(100, 100, 3)
```

Out[3]:

```
<matplotlib.image.AxesImage at 0x25c700eac48>
```



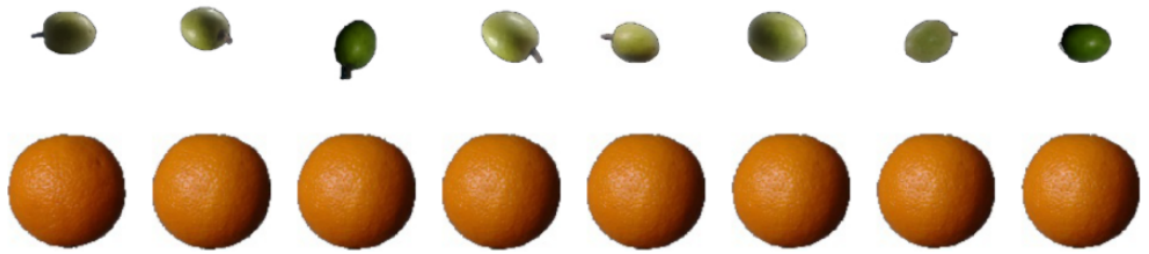
In [4]:
```
#view sample of training data class and shape of data class
train_veralu_dir = r"C:\Users\USER\Desktop\fruit_recognition\fruits-360\Training\Veralu"
number_veralu_train = len(os.listdir(train_veralu_dir))
print("Total training veralu images:", number_veralu_train)

#view sample of training data class and shape of data class
train_orange_dir = r"C:\Users\USER\Desktop\fruit_recognition\fruits-360\Training\Orange"
number_orange_train = len(os.listdir(train_orange_dir))
print("Total training Orange images:", number_orange_train)
```

```
Total training veralu images: 86
Total training Orange images: 479
```

In [5]:
```
veralu_names = os.listdir(train_veralu_dir)
veralu_names[:10]

orange_names = os.listdir(train_orange_dir)
orange_names[:10]
```

Out[5]:
```
['0_100.jpg',
 '100_100.jpg',
 '101_100.jpg',
 '102_100.jpg',
 '103_100.jpg',
 '104_100.jpg',
 '105_100.jpg',
 '106_100.jpg',
 '107_100.jpg',
 '108_100.jpg']
```

In [6]:
```
#view sample of training data class
nrows = 8
ncols = 8

pic_index = 0

fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)

pic_index+=8

veralu_pic = [os.path.join(train_veralu_dir,fname) for fname in veralu_names[pic_index-8:pic_index]]
orange_pic = [os.path.join(train_orange_dir,fname) for fname in orange_names[pic_index-8:pic_index]]

for i, img_path in enumerate(veralu_pic + orange_pic):
    sub = plt.subplot(nrows, ncols, i + 1)
    sub.axis("Off")

    img = mpimg.imread(img_path)
    plt.imshow(img)

plt.show()
```



Labels

Labels are the targets like in this project names of the fruits are labels.

Inputs

Inputs are the data that we feed into machine learning like in this project images are the inputs.

Training Data

We use training data when we train the models. We feed train data to machine learning and deep learning models so that model can learn from the data.

Validation Data

We use validation data while training the model. We use this data to evaluate the performance that how the model performs on training time.

Testing Data

We use testing data after training the model. We use this data to evaluate the performance that how the model performs after training. So, in this way first we get predictions from the trained model without giving the labels and then we compare the true labels with predictions and get the performance of the model.

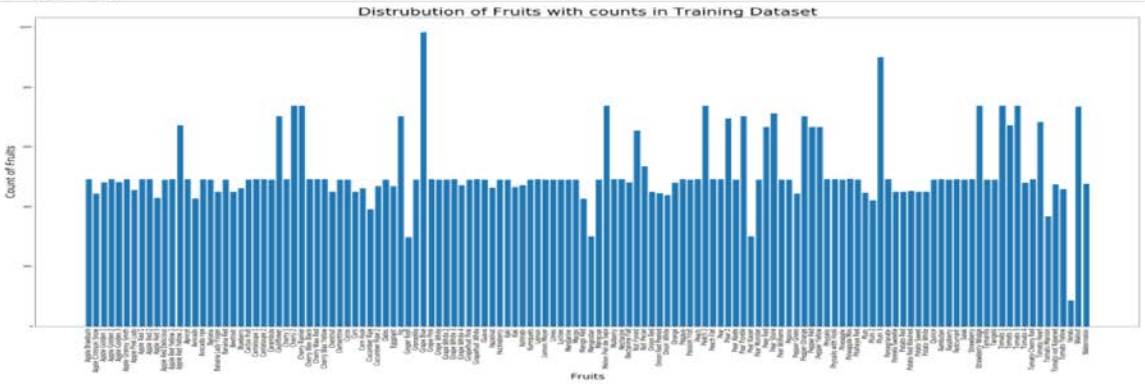
Training and Testing data Information

In [7]:
```
#view whole of training data and testing data
import os, os.path
train_categories = []
train_samples = []
for i in os.listdir(r"C:/Users/USER/Desktop/fruit_recognition/fruits-360/Training/"):
    train_categories.append(i)
    train_samples.append(len(os.listdir(r"C:/Users/USER/Desktop/fruit_recognition/fruits-360/Training/"+ i)))

test_categories = []
test_samples = []
for i in os.listdir(r"C:/Users/USER/Desktop/fruit_recognition/fruits-360/Test/"):
    test_categories.append(i)
    test_samples.append(len(os.listdir(r"C:/Users/USER/Desktop/fruit_recognition/fruits-360/Test/"+ i)))


print("Count of Fruits in Training data set:", sum(train_samples))
print("Count of Fruits in Testing data set:", sum(test_samples))
```

```
Count of Fruits in Training data set: 67778
Count of Fruits in Testing data set: 22774
```

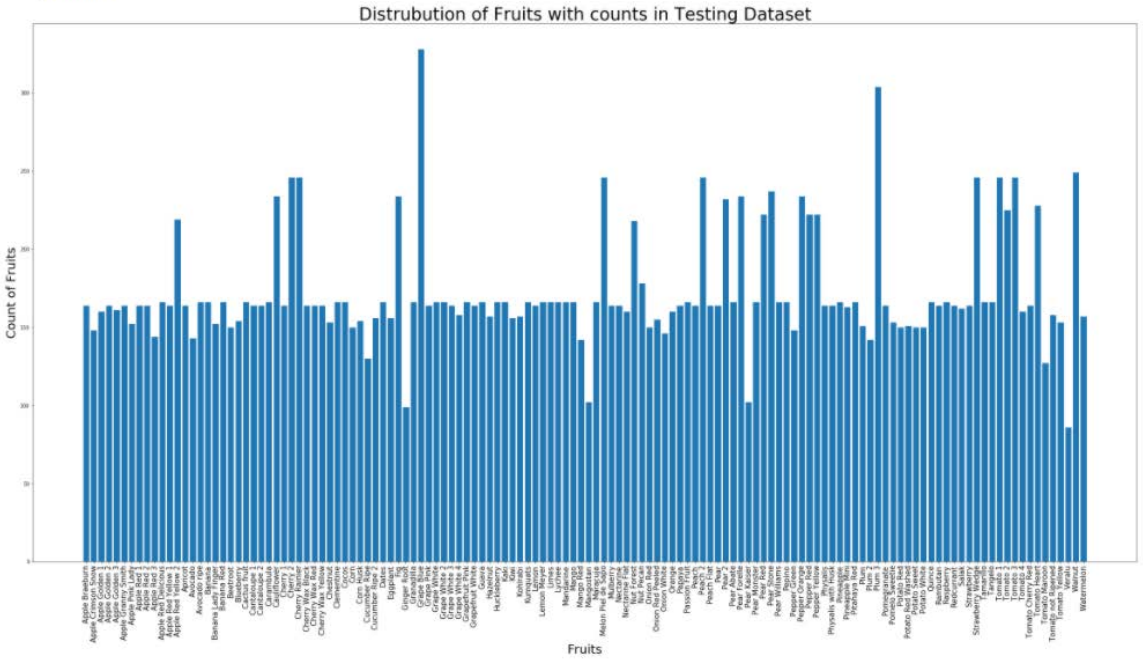
Distribution of Fruit with counts in Test Data Set

In [8]:
```
#distribution of training data
figure_size = plt.rcParams["figure.figsize"]
figure_size[0] = 40
figure_size[1] = 20
plt.rcParams["figure.figsize"] = figure_size
index = np.arange(len(train_categories))
plt.bar(index, train_samples)
plt.xlabel('Fruits', fontsize=25)
plt.ylabel('Count of Fruits', fontsize=25)
plt.xticks(index, train_categories, fontsize=15, rotation=90)
plt.title('Distrubution of Fruits with counts in Training Dataset', fontsize=35)
plt.show()
```



Distribution of Fruit with counts in Test Data Set

```
In [9]:
#distribution of testing data
index2 = np.arange(len(test_categories))
plt.bar(index2, test_samples)
plt.xlabel('Fruits', fontsize=25)
plt.ylabel('Count of Fruits', fontsize=25)
plt.xticks(index2, test_categories, fontsize=15, rotation=90)
plt.title('Distrubution of Fruits with counts in Testing Dataset', fontsize=35)
plt.show()
```



Loading the data and Splitting the dataset into training and testing with label names

We are loading all training and testing data

Saving inputs/images in x_train of training data and saving labels in y_train

Saving inputs/images in x_test of testing data and saving labels in y_test

Classes of Fruits

132 are labels/classes which are the names of fruits because we are using fruits of 132 types.

In [10]:
```
#datasets mapping to =======> number array
train_dir = r'C:/Users/USER/Desktop/fruit_recognition/fruits-360/Training/'
test_dir = r'C:/Users/USER/Desktop/fruit_recognition/fruits-360/Test/'

def load_dataset(data_path):
    data_loading = load_files(data_path)
    files_add = np.array(data_loading['filenames'])
    targets_fruits = np.array(data_loading['target'])
    target_labels_fruits = np.array(data_loading['target_names'])
    return files_add,targets_fruits,target_labels_fruits

x_train, y_train,target_labels = load_dataset(train_dir)
x_test, y_test,_ = load_dataset(test_dir)
```

In [11]:
```
no_of_classes = len(np.unique(y_train))
no_of_classes
```

Out[11]:
```
132
```

Vector of the y tarin first record

As we know that we are using 132 classes/labels, so we created a vector of 132 values.

We can see that there is only 1 and all are zero's that is showing a label of first image.

In [12]:
```
#view sample array data
y_train = np_utils.to_categorical(y_train,no_of_classes)
y_test = np_utils.to_categorical(y_test,no_of_classes)
y_train[0]
```

Out[12]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

Divide validation set into test and validation set

We are splitting the test data into validation data. Validation data will be used while training the model to check the performance during training and test data will be used after training the model.

In [13]:
```
#shape of validation data & testing data
x_test,x_valid = x_test[7000:],x_test[:7000]
y_test,y_vaild = y_test[7000:],y_test[:7000]
print('Vaildation X : ',x_valid.shape)
print('Vaildation y :',y_vaild.shape)
print('Test X : ',x_test.shape)
print('Test y : ',y_test.shape)
```

```
Vaildation X :  (7000,)
Vaildation y : (7000, 132)
Test X :  (15774,)
Test y :  (15774, 132)
```

Data Preprocessing

Till now the data is just images. We need to convert them into arrays form for the training and testing because machine learning model only understand the numeric data so we have to convert the images into arrays form.

In [14]:
```
#function of images are mapping array
def convert_image_to_array_form(files):
    images_array=[]
    for file in files:
        images_array.append(img_to_array(load_img(file)))
    return images_array

x_train = np.array(convert_image_to_array_form(x_train))
print('Training dataset shape : ',x_train.shape)

x_valid = np.array(convert_image_to_array_form(x_valid))
print('Validation dataset shape : ',x_valid.shape)

x_test = np.array(convert_image_to_array_form(x_test))
print('Test dataset shape : ',x_test.shape)

print('initial training image shape ',x_train[0].shape)

```

```
Training dataset shape :  (67778, 100, 100, 3)
Validation dataset shape :  (7000, 100, 100, 3)
Test dataset shape :  (15774, 100, 100, 3)
initial training image shape  (100, 100, 3)
```

Features scaling from 0-255 to 0-1

After converting the images into arrays form, the features/pixels ranges are from 0 to 255. We are scaling the features/pixels from 0-255 to 0-1 range.

Why we are doing this?

By doing this, we can reduce the training time. Because we can also feed the same features but when the model will train so it takes more time by calculating the values from 0-255 than 0-1. That is why we are scaling the features.

In [15]:
```
#preprocessing data ===> scale
x_train = x_train.astype('float32')/255
x_valid = x_valid.astype('float32')/255
x_test = x_test.astype('float32')/255
```

Implementing TensorFlow based model for Training

Step 1

- We are calling base Sequential model for training and for further tuning of parameters on image data. We must call it when we work on the keras, tensorflow based libraries.

Step 2

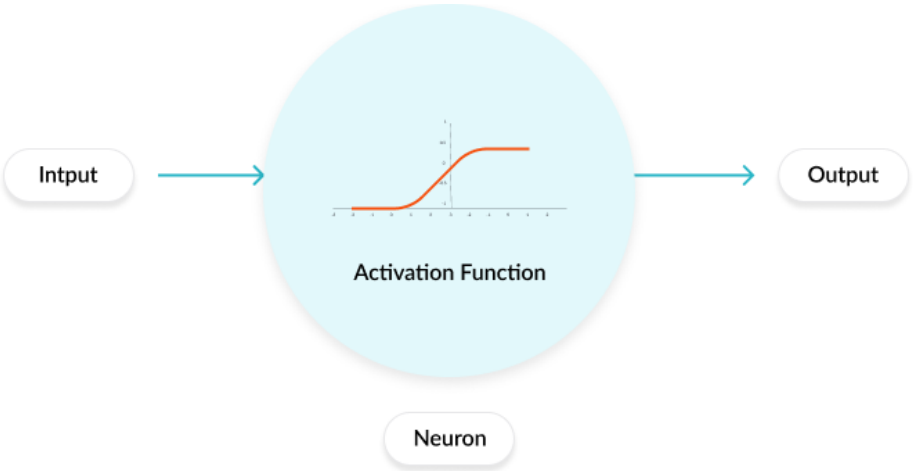
- Conv2D is 2D convolutional layer (where filters are applied to original image with specific features map to reduce the number of features), Conv2D layer create the convolution kernel (fixed size of boxes to apply on the image like below in the example gif) that take input of 16 filters which help to produce a tensor of outputs. We are giving input of the image with size of 100 width and 100 height and 3 is the channel for RGB.

Step 3

- Activation function is node that is put at the end of all layers of neural network model or in between neural network layers. Activation function help to decide which neuron should be pass

and which neuron should fire. So, activation function of node defines the output of that node given an input or set of inputs.

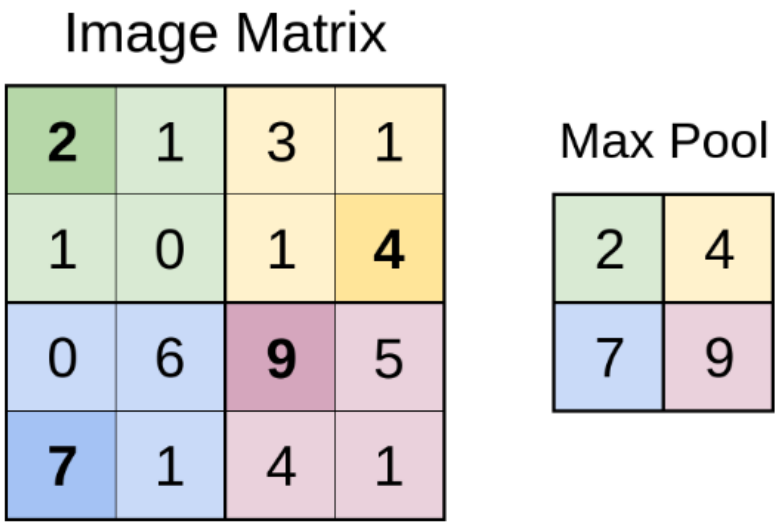
Activation function image

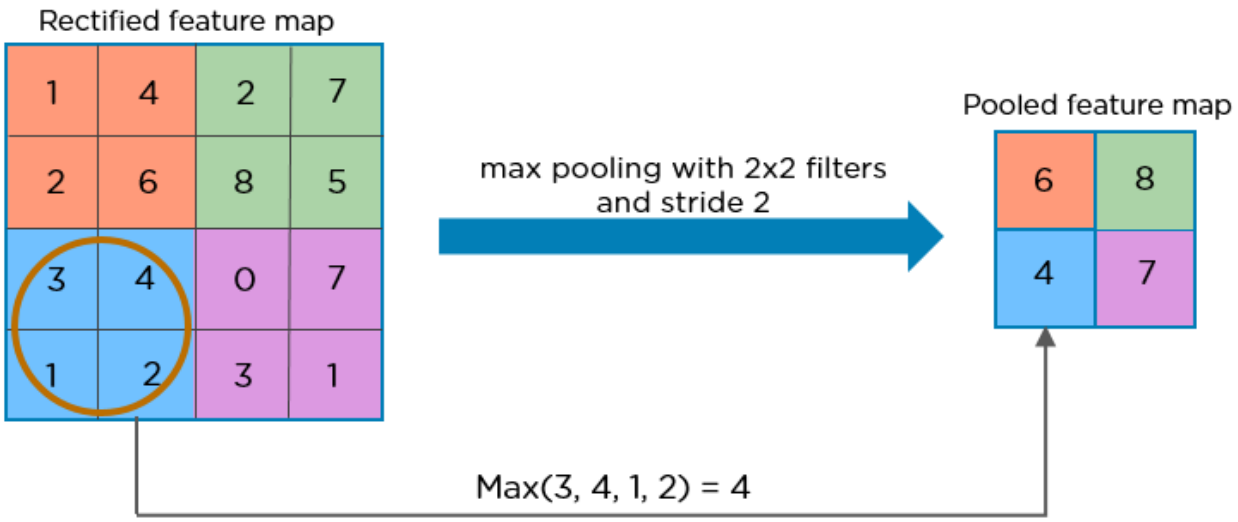


Step 4

- Maxpooling is a pooling operation that calculates maximum value in each patch of each feature map. It takes the value from input vectors and prepare the vector for the next layers.

Max pooling operation





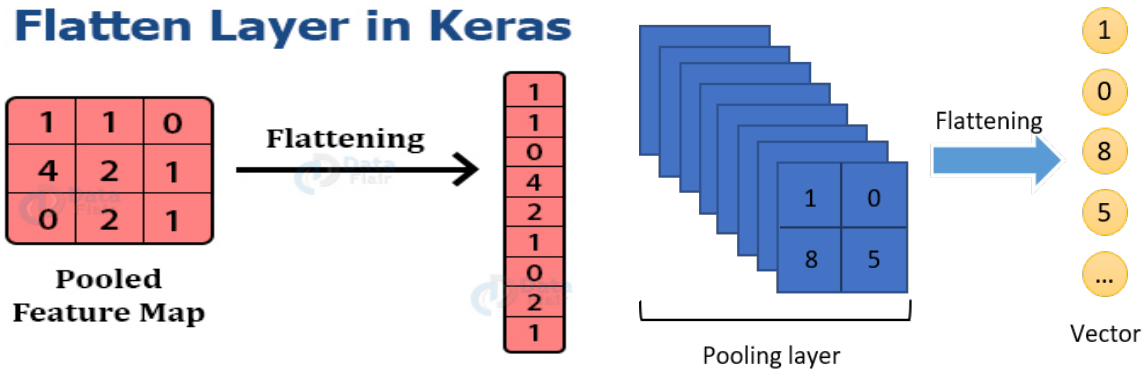
Step 5

- Dropout layer drop some neurons from previous layers. why we apply this? We apply this to avoid the overfitting problems. In overfitting, model give good accuracy on training time but not good on testing time

Step 6

- Flatten layer convert the 2D array into 1D array of all features.

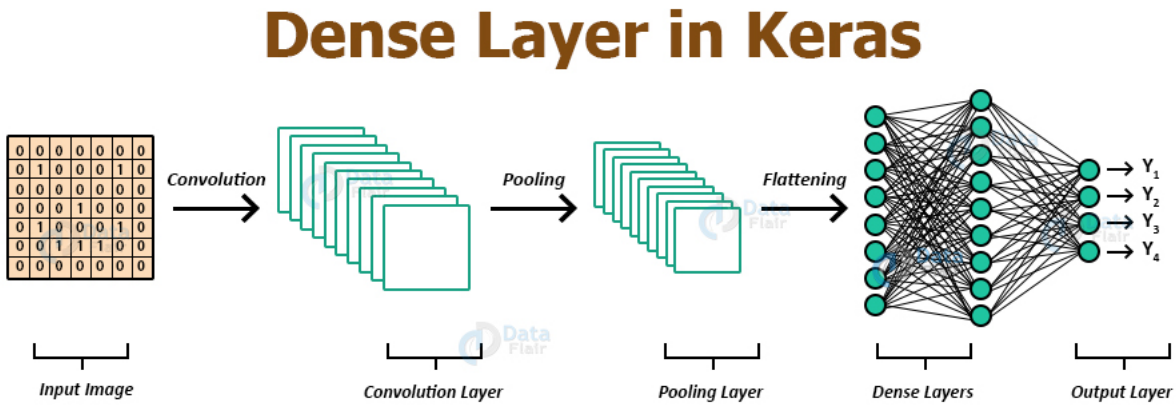
Flatten layer image



Step 7

- Dense layer reduces the outputs by getting inputs from Flatten layer. Dense layer uses all the inputs of previous layer neurons and perform calculations and send 150 outputs.

Dense layer



CNN model

In [16]:

```
#define our network model
def tensorflow_based_model():
    # step 1
    model = Sequential()
    # step2
    model.add(Conv2D(filters = 16, kernel_size = 2,input_shape=(100,100,3),padding='same'))
    # step3
    model.add(Activation('relu'))
    # step4
    model.add(MaxPooling2D(pool_size=2))
    # repeating step 2 and step3 but with more filters of 32
    model.add(Conv2D(filters = 32,kernel_size = 2,activation= 'relu',padding='same'))
    # repeating step 4 again
    model.add(MaxPooling2D(pool_size=2))
    # repeating step 2 and step3 but with more filters of 64
    model.add(Conv2D(filters = 64,kernel_size = 2,activation= 'relu',padding='same'))
    # repeating step 4 again
    model.add(MaxPooling2D(pool_size=2))
    # repeating step 2 and step3 but with more filters of 64
    model.add(Conv2D(filters = 128,kernel_size = 2,activation= 'relu',padding='same'))
    # repeating step 4 again
    model.add(MaxPooling2D(pool_size=2))
    # step5
    model.add(Dropout(0.3))
    # step 6
    model.add(Flatten())
    # step 7
    model.add(Dense(150))
    # setp 3
    model.add(Activation('relu'))
    # step 5
    model.add(Dropout(0.4))
    # setp3 and step7. but this time, we are using activation function as softmax
    # (if we train on two classes then we set sigmoid)
    model.add(Dense(132,activation = 'softmax'))
    # function returning the value when we call it
    return model
```

Model Compilation

First, we are calling the model

We are using 132 classes so we set loss as "categorical_crossentropy". We use loss as "binary_crossentropy" for two classes.

Optimizer is a function that used to change the features of neural network such as learning rate (how the model learns with features) in order to reduce the losses. So, the learning rate of neural network to reduce the losses is defined by optimizer.

We are setting metrics=accuracy because we are going to calculate the percentage of correct predictions over all predictions on the validation set

In [17]:
```
# here we are calling the function of created model
model = tensorflow_based_model()
# here we are get the summary of created model
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 100, 100, 16)      208
_________________________________________________________________
activation (Activation)      (None, 100, 100, 16)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 50, 50, 32)        2080
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 64)        8256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 128)       32896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout (Dropout)            (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 150)               691350
_________________________________________________________________
activation_1 (Activation)    (None, 150)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_1 (Dense)              (None, 132)               19932
=================================================================
Total params: 754,722
Trainable params: 754,722
Non-trainable params: 0
_________________________________________________________________
```

Training model with Parameter tuning

We are feeding the training data and validation data to start training of model.

We set the following parameters:

Batch size =32 so the model takes 80 images in each iteration and train them. Batch size is a term used in machine learning and refers to the number of training examples utilized in one iteration.

Epochs =30 so the model will train on the data 30 times. Epoch is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed.

We can choose batch_size, and epochs as we want so the good practice is to set some values and train the model if the model will not give the good results, we can change it and then try again for the training of the model. We can repeat this process many time until we will not get the good results and this process called as parameter tuning.

```
#model training process and saving process
history = model.fit(x_train,y_train,
        batch_size = 32,
        epochs = 30,
        validation_data=(x_valid, y_vaild),
        verbose=2,
        shuffle=True
)

filepath = r"C:\Users\USER\Desktop\fruit_recognition\fruits-360\model"
tf.keras.models.save_model(
    model,
    filepath,
    overwrite=True,
    include_optimizer=True,
    save_format="tf",
    signatures=None
)
model.save("fruit_classification_model.h5")
```

```
Train on 67692 samples, validate on 7000 samples
Epoch 1/30
67692/67692 - 157s - loss: 1.1449 - accuracy: 0.6860 - val_loss: 0.3172 - val_accuracy: 0.9047
Epoch 2/30
67692/67692 - 150s - loss: 0.1571 - accuracy: 0.9486 - val_loss: 0.1818 - val_accuracy: 0.9621
Epoch 3/30
67692/67692 - 151s - loss: 0.0986 - accuracy: 0.9684 - val_loss: 0.1675 - val_accuracy: 0.9649
Epoch 4/30
67692/67692 - 180s - loss: 0.0795 - accuracy: 0.9763 - val_loss: 0.1222 - val_accuracy: 0.9754
Epoch 5/30
67692/67692 - 154s - loss: 0.0713 - accuracy: 0.9799 - val_loss: 0.1210 - val_accuracy: 0.9801
Epoch 6/30
```

After some steps

```
67692/67692 - 163s - loss: 0.2120 - accuracy: 0.9820 - val_loss: 1.1134 - val_accuracy: 0.9787
Epoch 26/30
67692/67692 - 161s - loss: 0.2404 - accuracy: 0.9827 - val_loss: 0.8605 - val_accuracy: 0.9784
Epoch 27/30
67692/67692 - 160s - loss: 0.2394 - accuracy: 0.9836 - val_loss: 0.5530 - val_accuracy: 0.9836
Epoch 28/30
67692/67692 - 156s - loss: 0.2386 - accuracy: 0.9845 - val_loss: 1.0656 - val_accuracy: 0.9800
Epoch 29/30
67692/67692 - 154s - loss: 0.2565 - accuracy: 0.9839 - val_loss: 0.4124 - val_accuracy: 0.9716
Epoch 30/30
67692/67692 - 157s - loss: 0.2576 - accuracy: 0.9839 - val_loss: 0.7715 - val_accuracy: 0.9803
```

We need to do all the above configurations to train the model. If we will not set settings correctly then we could not get the desired results

Accuracy Score on the test data

Accuracy is the number of correctly recognized images from all the images.

For example, if the trained model recognizes the 90 images correct and 10 images wrong from total of 100 images then the accuracy score will be 90%.

Accuracy= Total number of correct predictions/Total number of predictions

In [19]:
```
#testing accuracy of trained model
acc_score = model.evaluate(x_test, y_test) #we are starting to test the model here
print('\n', 'Test accuracy:', acc_score[1])
```

```
15774/15774 [==============================] - 31s 2ms/sample - loss: 0.6925 - accuracy: 0.9781

 Test accuracy: 0.9780652
```

Visualization with Predictions

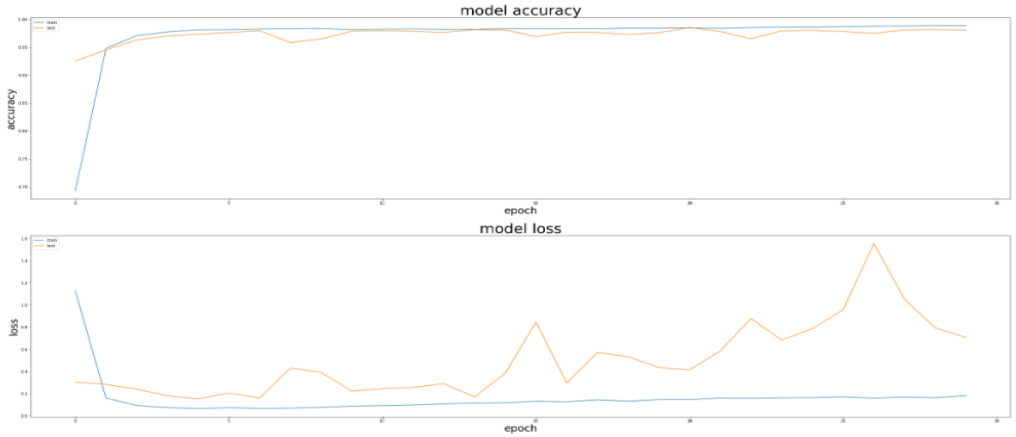
We are using the trained model and getting predictions on the test data

Outside the bracket are the predictions names of the fruits

Inside the bracket are the true label names of the fruits

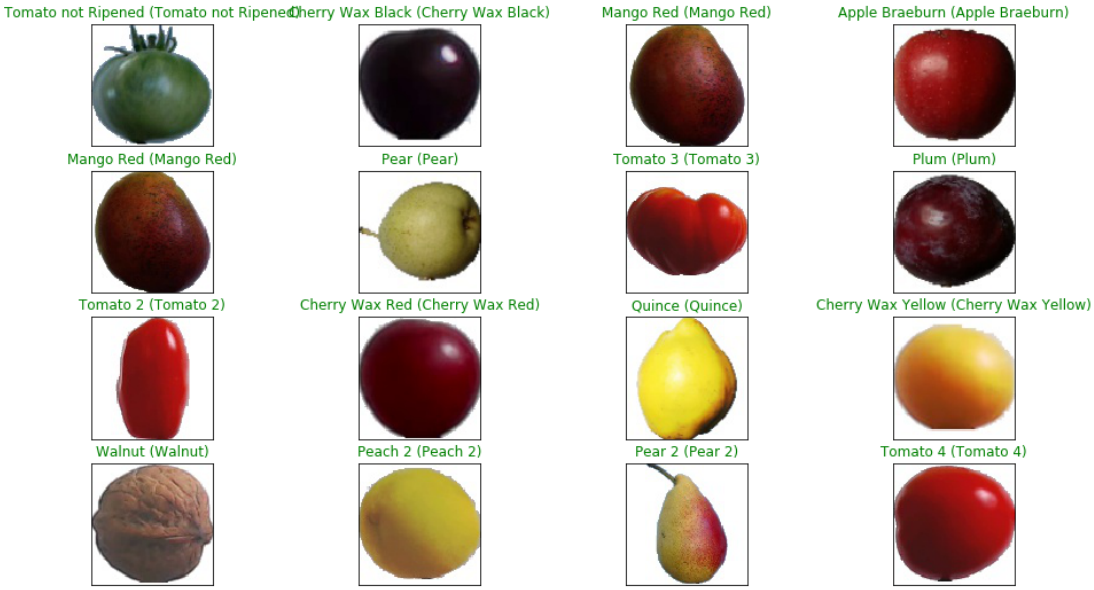
In here red colored fruits were deviated from the trained model and green labels shows successful predictions in accurate model

In [21]:
```
#how the model accuracy goes on and loss goes on
plt.figure(1)
plt.subplot(211)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy',fontsize=35)
plt.ylabel('accuracy',fontsize=25)
plt.xlabel('epoch',fontsize=25)
plt.legend(['train', 'test'], loc='upper left')

plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss',fontsize=35)
plt.ylabel('loss',fontsize=25)
plt.xlabel('epoch',fontsize=25)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [20]:

```
#here we check model predictions
predictions = model.predict(x_test)
fig = plt.figure(figsize=(16, 9))
for i, idx in enumerate(np.random.choice(x_test.shape[0], size=16, replace=False)):
    ax = fig.add_subplot(4, 4, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    pred_idx = np.argmax(predictions[idx])
    true_idx = np.argmax(y_test[idx])
    ax.set_title("{} ({})".format(target_labels[pred_idx], target_labels[true_idx]),
                 color=("green" if pred_idx == true_idx else "red"))
```



Visualization the loss and accuracy with respect to epochs

We are looking at the history of the model of each epoch as we trained our model on 30 epochs.

Blue line shows the training accuracy and also the training loss.

Orange line shows the Testing accuracy and also the testing loss.

Accuracy and loss on the train and test data start from zero and finally close to 1 (100%).

```
plt.figure(1)
plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In here model accuracy reached 97.4 precent.

Save Classification model

Code line 18

```
filepath = r"C:\Users\USER\Desktop\fruit_recognition\fruits-360\model"
tf.keras.models.save_model(
    model,
    filepath,
    overwrite=True,
    include_optimizer=True,
    save_format="tf",
    signatures=None
)
model.save("fruit.h5")
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow_core\python\ops\resource_variable_ops.py:1786: ca
lling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will b
e removed in a future version.
output; double click to hide dating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: C:\Users\USER\Desktop\fruit_recognition\fruits-360\model\assets
```

In here successfully trained model were saved in provided root dictionary.

Conclusions and further work

First, we were described a complex database of images with fruits with including Sri Lankan endemic fruits. Also, we were made some numerical approximations In order to the images according to their content by using TensorFlow library. From start to end, our point of view one of the main objectives for the future is to improve the accuracy of the neural network as well as improving best fitted model for recognized problem. This involves further experimenting with the structure of the network. In here we can organize our task as follows

We used the fruits360 dataset and some of the fruits we were added to it (e.g.: Veralu); explored the data with different ways.

We were prepared the image data and scale the exact manner to extract the features.

We were trained model based on TensorFlow with all settings till achieved better accuracy.

We were evaluated the model with accuracy and look at the performance of the model with plots.

In this image-based project, we were prepared the most data of images from internet and some of images from our group members. Much larger data set leads to much high accuracy on model.

We were called specifically on this classification problem as multi class classification because we were using in total 132 classes of fruits. When the model is finished, the accuracy was approximately about 97%. There are 132 variety of fruit classes (Not only endemic to Sri Lanka) and 4 classes(fruits) were deviated from trained model and also happy to say 128 classes(fruits) worked properly. Can be used for

future works such that can develop a real time mobile application for Foreign and local travelers. Also, there are some of the fruits we were not covered in our research because of the deficit of some images data. They will be included in the model in the future. Thanks, in advanced who help to achieve our goals in this project.

Appendix

In this section we present the source code and project structure used in the numerical experiment described in this paper. The source code can be downloaded from GitHub [38].

References

[1] Bargoti, S., and Underwood, J. Deep fruit detection in orchards. In 2017 IEEE International Conference on Robotics and Automation (ICRA)(May 2017), pp. 3626–3633.)4

[2] Barth, R., IJsselmuiden, J., Hemming, J., and Henten, E. V. Data synthesis methods for semantic segmentation in agriculture: A capsicum annuum dataset. Computers and Electronics in Agriculture 144 (2018),284 – 296.)4

[3] Chan, T. F., and Vese, L. A. Active contours without edges. IEEE Transactions on Image Processing 10, 2 (Feb 2001), 266–277.)5

[4] Cheng, H.,Damerow, L., Sun,Y., and Blanke, M. Early yield prediction using image analysis of apple fruit and tree canopy features with neural networks. Journal of Imaging 3, 1 (2017).)4

[5] Cires¸an, D. C., Giusti, A., Gambardella, L. M., and Schmidhuber, J. Deep neural networks segment neuronal membranes in electron microscopy images. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2 (USA, 2012), NIPS'12, Curran Associates Inc., pp. 2843–2851.)5

[6] Cires¸an, D. C.,Meier, U.,Masci, J., Gambardella, L. M., and Schmid- huber, J. Flexible, high performance convolutional neural networks for image classification. In Proceedings of the Twenty Second International Joint Conference on Artificial Intelligence - Volume Volume Two (2011), IJCAI'11, AAAI Press, pp. 1237–1242.)5

[7] Ciresan, D. C., Meier, U., and Schmidhuber, J. Multi-column deep neural networks for image classification. CoRR abs/1202.2745 (2012).)6

[8] Clevert, D., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). CoRR abs/1511.07289 (2015).)27

[9] Hannun, A. Y., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. Deep speech: Scaling up end-to-end speech recognition. CoRR abs/1412.5567 (2014).)15

[10] Hemming, J., Ruizendaal, J.,Hofstee, J.W., and vanHenten, E. J. Fruit detectability analysis for di_erent camera positions in sweet-pepper. Sensors 14, 4 (2014), 6032–6044.)4

[11] Kapach, K., Barnea, E., Mairon, R., Edan, Y., and Ben-Shahar, O. Computer vision for fruit harvesting robots – state of the art andchallenges ahead. Int. J. Comput. Vision Robot. 3, 1/2 (Apr. 2012), 4 34.)4

[12] Krizhevsky, A., Nair, V., and Hinton, G. The cifar dataset. [Online; accessed 27.10.2018].)2, 6

[13] LeCun, Y., Cortes, C., and Burges, C. J. The mnist database of handwritten digits. [Online; accessed 27.10.2018].)6

[14] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Proceedings of the 26th Annual International Conference on Machine Learning (New York, NY, USA, 2009), ICML '09, ACM, pp. 609–616.)8
[15] Li, D., Zhao, H., Zhao, X., Gao, Q., and Xu, L. Cucumber detection based on texture and color in greenhouse. International Journal of Pattern Recognition and Artificial Intelligence 31 (01 2017).)4
[16] Liang, M., and Hu, X. Recurrent convolutional neural network for object recognition. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2015), pp. 3367–3375.)6, 8
[17] Mumford, D., and Shah, J. Optimal approximations by piecewise smooth functions and associated variational problems. Communications on Pure and Applied Mathematics 42, 5 (1989), 577–685.)5
[18] Ninawe, P., and Pandey, M. S. A completion on fruit recognition system using k-nearest neighbors algorithm. In International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) (2014), vol. 3.)5
[19] O'Boyle, B., and Hall, C. What is google lens and how do you use it? [Online; accessed 05.05.2018].)2
[20] Oltean, M., and Muresan, H. Fruits 360 dataset on github. [Online; accessed 27.10.2018].)1, 10, 28
[21] Oltean, M., and Muresan, H. Fruits 360 dataset on kaggle. [Online; accessed 27.10.2018].)1, 10
[22] Puttemans, S., Vanbrabant, Y., Tits, L., and Goedem, T. Automated visual fruit detection for harvest estimation and robotic harvesting. In 2016 Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA) (Dec 2016), pp. 1–6.)4
[23] Rahnemoonfar, M., and Sheppard, C. Deep count: Fruit counting based on deep simulated learning. Sensors 17, 4 (2017).)4
[24] Ren, S., He, K., Girshick, R. B., and Sun, J. Faster R-CNN: towards real-time object detection with region proposal networks. CoRRabs/1506.01497 (2015).)4
[25] Sa, I., Ge, Z., Dayoub, F., Upcroft, B., Perez, T., andMcCool, C. Deepfruits: A fruit detection system using deep neural networks. Sensors 16, 8 (2016).)3, 4
[26] Schmidhuber, J. Deep learning in neural networks:Anoverview. CoRR abs/1404.7828 (2014).)5
[27] Selvaraj, A., Shebiah, N., Nidhyananthan, S., and Ganesan, L. Fruit recognition using color and texture features. Journal of Emerging Trends in Computing and Information Sciences 1 (10 2010), 90–94.)4
[28] Song, Y., Glasbey, C., Horgan, G., Polder, G., Dieleman, J., and van der Heijden, G. Automatic fruit recognition and counting from multiple images. Biosystems Engineering 118 (2014), 203 – 215.)3
[29] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. Striving for simplicity: The all-convolutional net. CoRR abs/1412.6806 (2014).)6, 27
[30] Srivastava, R. K., Greff, K., and Schmidhuber, J. Training very deep networks. CoRR abs/1507.06228 (2015).)5
[31] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. CoRR abs/1409.4842 (2014).)15
[32] TensorFlow. Tensorflow. [Online; accessed 05.05.2018].)3, 15
[33] Wikipedia. Convolution in mathematics. [Online; accessed 05.05.2018].)6
[34] Wikipedia. Deep learning article on wikipedia. [Online; accessed 05.05.2018].)5
[35] Wikipedia. Google lens on wikipedia. [Online; accessed 05.05.2018].)2
[36] Xiong, J., Liu, Z., Lin, R., Bu, R., He, Z., Yang, Z., and Liang, C. Green grape detection and picking-point calculation in a night-time natural environment using a charge-coupled device (ccd) vision sensor with artificial illumination. Sensors 18, 4 (2018).)5
[37] Zawbaa, H.,Abbass, M.,Hazman, M., andHassanien, A. E. Automatic fruit image recognition system based on shape and color features. Communications in Computer and Information Science 488 (11 2014), 278– 290.)4
[38] https://github.com/buddikatdk/Machine_Learning_Group_Project.git