Machine Learning Engineer Nanodegree

Capstone Project

Sahel Mastoureshgh
July 14th, 2018

I. Definition

Project Overview

This project focus on fruits image classification and object recognition task by using deep learning and convolutional neural networks.

The purpose of this project is to develop software solutions to analyzing the fruit images
captured to improve the accuracy of current measurements of dietary consumption.

For the dataset, I obtained a large data set of images of fruits from a Kaggle dataset called Fruits 360 dataset. There are 74 types of fruits in this dataset

Total number of images were originally 49606 which I reduced it to 10766.

There were pervious attempts to use neural network and deep learning in area of fruit image recognition. In paper [1], they used deep neural network that used by autonomous robot for harvesting fruit. They used RGB and NIR (near infra-red) images to train their model. They introduced two methods, late and early fusion, that use the combination of RGB and NIR imagery models. Early fusion, alters the structure of the input layer of the VGG network so that the input data layer has four channels (3 channels from RGB and 1 channel from NIR), Late fusion uses two independently trained models that are merged by obtaining prediction from both models and averaging the result. The result is multi model network which has better performance than existing networks.

Problem Statement

We investigate a problem of fruits classification. Given an image, we want to find if the image contains fruit or not and what type of fruit it is.

My intended solution for above problem is to use a machine learning model, specifically Convolutional Neural Networks for image classification.

The strategy I will use to achieve this is to download and preprocess the Kaggle fruits data. Create a bench mark using plain vanilla CNN, Also, I will use transfer learning to create a CNN using Xception pre-computed the features that are currently available in Keras.

finally, I will accept any user-supplied image as input. If a fruit is detected in the image, then it will provide an estimate of the type of fruits.

For fruit detection, I will use transfer learning CNN using Xception trained based on ImageNet which contains over 10 million images containing an object from one of 1000 categories.
For classifying image type, I will use the same approach as detecting fruit image but the last convolutional output of Xception is fed to the new model which has a global average pooling layer and a dense layer with SoftMax activation.

Metrics

For fruit image multi-class classification task, we used accuracy metric to evaluate the quality of our model. accuracy metric indicates how frequently the result output matches the correct label. To maximize accuracy of our predictions, we will use a loss function which evaluates how wrong our model predictions are and then it will propagate those errors back via a process called backpropagation which is a technique for circulating the errors returned by the loss function back to the network.

Because our problem is a classification problems, we used accuracy with loss function Categorical Cross-entropy which is used for type of problem related to classification between many categories from a known set of categories.

For optimizer CNN model, I used Nadam and RMSprop. I compared the result of these two optimizer and picked the one which has better accuracy.
I chose these two optimizers based on Tenser flow websitehttps://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Nadam.

Optimizer	Accuracy
nadam	90.47%
rmsprop	89.77%

II. Analysis

Data Exploration

The dataset is obtained from set of images of fruits in Kaggle dataset, there is 74 types

of fruits. The image size is 100*100 pixels with white background. The data set of images obtained from Kaggle has been reduced to 10766 images

Training	6563
Validation	2218
Testing	1985

Below table shows the number of training image data set in each folder of fruit label.

	number of images in Training
Apple Braeburn	124.0
Apple Golden 1	103.0
Apple Golden 2	103.0
Apple Golden 3	103.0
Apple Granny Smith	103.0
Apple Red 1	103.0
Apple Red 2	103.0
Apple Red 3	85.0
Apple Red Delicious	75.0
Apple Red Yellow	103.0
Apricot	103.0
Avocado ripe	82.0
Avocado	45.0
Banana Red	89.0
Banana	53.0
Cactus fruit	100.0
Cantaloupe 1	51.0
Cantaloupe 2	67.0
Carambula	79.0
Cherry 1	103.0
Cherry 2	104.0
Cherry Rainier	65.0
Cherry Wax Black	79.0
Cherry Wax Red	70.0
Cherry Wax Yellow	76.0
Clementine	75.0
Cocos	100.0
Dates	78.0
Granadilla	87.0
Grape Pink	103.0
Grape White 2	106.0
Grape White	101.0
Grapefruit Pink	97.0
Grapefruit White	103.0
Guava	93.0
Huckleberry	83.0
Kaki	76.0
Kiwi	103.0
Kumquats	74.0
Lemon Meyer	103.0
Lemon	76.0
Limes	48.0
Lychee	103.0
Mandarine	94.0
Mango	66.0
Maracuja	54.0
Melon Piel de Sapo	109.0
Mulberry	84.0
Nectarine	103.0
Orange	103.0
Papaya	103.0
Passion Fruit	44.0
Peach Flat	103.0
Peach	103.0
Pear Abate	103.0
Pear Monster	104.0
Pear Williams	84.0
Pear	104.0
Pepino	103.0
Physalis with Husk	96.0
Physalis	88.0
Pineapple Mini	92.0
Pineapple	88.0
Pitahaya Red	88.0
Plum	103.0
Pomegranate	103.0
Quince	80.0
Rambutan	44.0
Raspberry	96.0
Salak	72.0
Strawberry Wedge	103.0
Strawberry	70.0
Tamarillo	109.0
Tangelo	87.0

Exploratory Visualization

Images in the dataset are standard RGB images 100 by 100 pixels with white background. each fruit category has images of fruits from different angel. The training, validation and test datasets have 74 categories of fruits each have several images. Table below show sample of data which have been used.

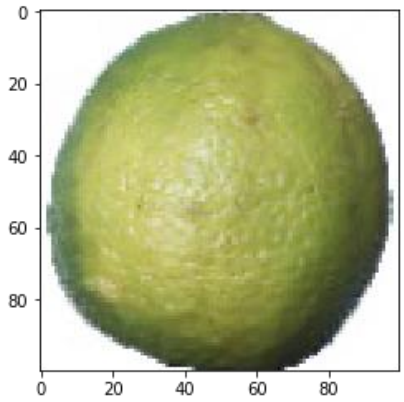
image label	Training images
Banana	
Apple Granny Smith	
Mango	
Orange	

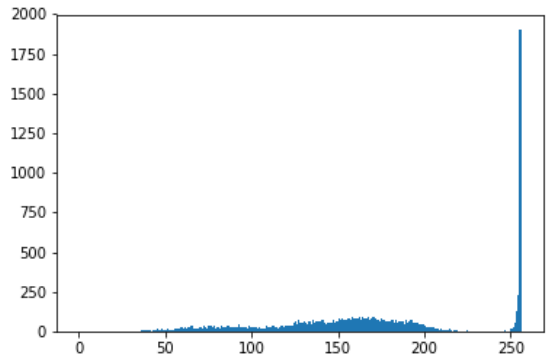
Table below shows the label of fruits type in our dataset and number of training, validation and test images in our dataset for each fruit type

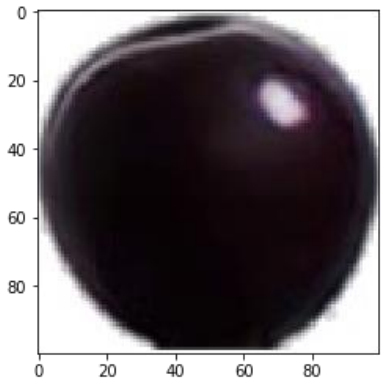
	label	number of images in Training	number of images in Validation
Apple Braeburn	124.0	28.0	7.0
Apple Golden 1	103.0	28.0	7.0
Apple Golden 2	103.0	28.0	7.0
Apple Golden 3	103.0	29.0	7.0
Apple Granny Smith	103.0	28.0	7.0
Apple Red 1	103.0	28.0	7.0
Apple Red 2	103.0	28.0	7.0
Apple Red 3	85.0	16.0	6.0
Apple Red Delicious	75.0	23.0	41.0
Apple Red Yellow	103.0	28.0	7.0
Apricot	103.0	28.0	7.0
Avocado ripe	82.0	15.0	6.0
Avocado	45.0	14.0	65.0
Banana Red	89.0	22.0	34.0
Banana	53.0	15.0	41.0
Cactus fruit	100.0	61.0	13.0
Cantaloupe 1	51.0	12.0	59.0
Cantaloupe 2	67.0	39.0	43.0
Carambula	79.0	35.0	31.0
Cherry 1	103.0	28.0	7.0
Cherry 2	104.0	30.0	91.0
Cherry Rainier	65.0	28.0	127.0
Cherry Wax Black	79.0	45.0	31.0
Cherry Wax Red	70.0	7.0	24.0
Cherry Wax Yellow	76.0	54.0	14.0
Clementine	75.0	7.0	35.0
Cocos	100.0	50.0	16.0
Dates	78.0	21.0	50.0
Granadilla	87.0	13.0	23.0
Grape Pink	103.0	28.0	7.0
Grape White 2	106.0	41.0	4.0
Grape White	101.0	45.0	15.0
Grapefruit Pink	97.0	42.0	13.0
Grapefruit White	103.0	28.0	7.0
Guava	93.0	30.0	17.0
Huckleberry	83.0	13.0	28.0
Kaki	76.0	46.0	37.0
Kiwi	103.0	27.0	7.0
Kumquats	74.0	42.0	37.0
Lemon Meyer	103.0	28.0	7.0
Lemon	76.0	44.0	35.0
Limes	48.0	21.0	62.0
Lychee	103.0	24.0	11.0
Mandarine	94.0	67.0	16.0
Mango	66.0	13.0	48.0
Maracuja	54.0	14.0	66.0
Melon Piel de Sapo	109.0	66.0	93.0
Mulberry	84.0	26.0	27.0
Nectarine	103.0	28.0	7.0
Orange	103.0	30.0	7.0
Papaya	103.0	28.0	7.0
Passion Fruit	44.0	16.0	66.0
Peach Flat	103.0	28.0	7.0
Peach	103.0	28.0	7.0
Pear Abate	103.0	28.0	7.0
Pear Monster	104.0	70.0	6.0
Pear Williams	84.0	17.0	26.0
Pear	104.0	65.0	6.0
Pepino	103.0	35.0	8.0
Physalis with Husk	96.0	23.0	21.0
Physalis	88.0	35.0	31.0
Pineapple Mini	92.0	44.0	24.0
Pineapple	88.0	33.0	24.0
Pitahaya Red	88.0	32.0	22.0
Plum	103.0	22.0	7.0
Pomegranate	103.0	28.0	7.0
Quince	80.0	18.0	40.0
Rambutan	44.0	12.0	66.0
Raspberry	96.0	33.0	14.0
Salak	72.0	31.0	48.0
Strawberry Wedge	103.0	28.0	7.0
Strawberry	70.0	27.0	136.0
Tamarillo	109.0	39.0	4.0
Tangelo	87.0	7.0	23.0

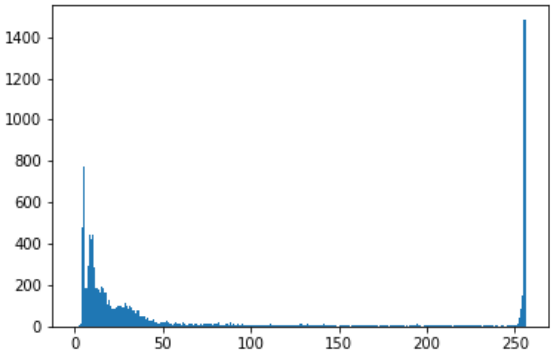
Intensity histogram which gives an overall idea about the intensity distribution of an image. It is a plot with pixel values in X-axis and corresponding number of pixels in the image on Y-axis.

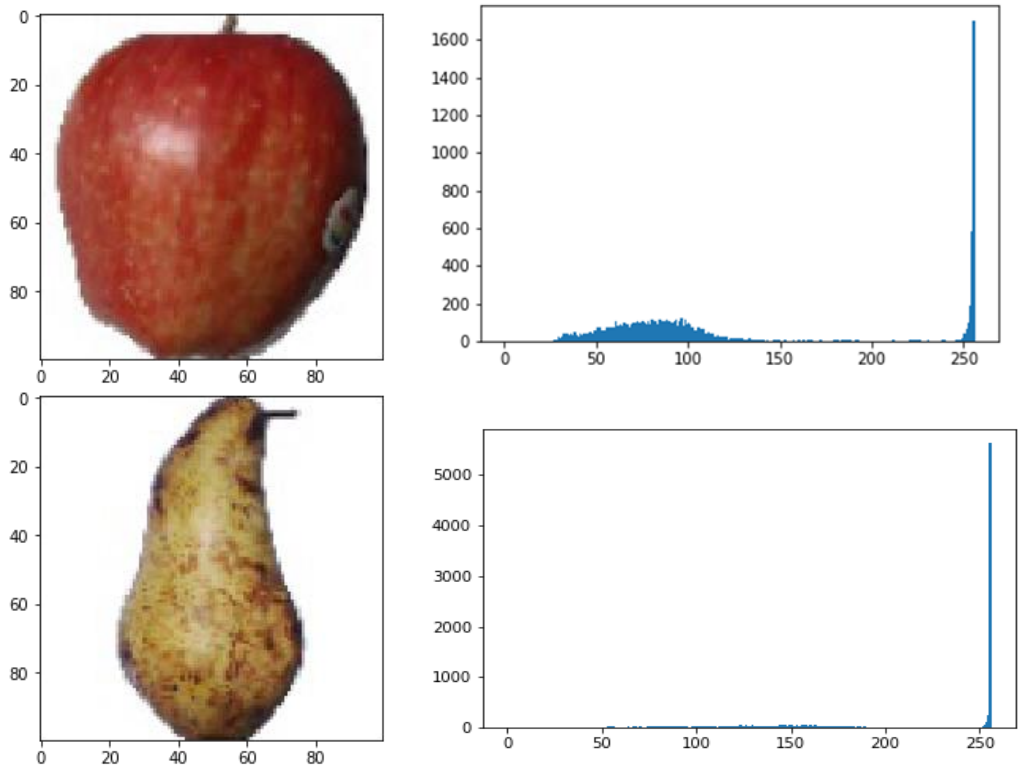
Below I put some example of my training data set Intensity histogram











Algorithms and Techniques

I applied a Convolution Neural Networks on the image data set for bench mark model and Convolution Neural Networks transform learning for actual solution to fruit image classification problem.

I explain first Convolution Neural Networks then Convolution Neural Networks transform learning

Convolution Neural Networks (CNNs). It is a family of deep learning architectures that use the convolution operation of image processing to extract features from the images that can explain the object that we want to classify.
A typical CNN architecture consists of multiple layers that do different tasks. Below I explain each layer

- Input layer: This is the first layer in any CNN architecture. All the subsequent convolution and pooling layers expect the input to be in a specific format. The input to a convolutional layer is a m x m x r image where m is the height and width of the image and r is the number of channels
- convolutional layer: The main purpose of this layer is to extract features from the input images and then feed them to a linear classifier. The whole idea of stacking a bunch of convolution layers is to be able to detect image features such as edges anywhere in the image. the convolutional layer will have k filters of size n x n x q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller. The filter argument which specifies the number of filters to be applied to the image where the higher the number of filters, the more features are extracted from the input image. The padding parameters specifies which we used "same" here which will result in the output has the same length as the original input. We also have the activation function that should be used for the output of the convolution operation.
- Pooling layer: This layer is mainly for reducing the dimensionality of the output of the convolution layer while keeping the important information in the newly reduced version. We used max pooling which takes the largest value from one patch of an image, places it in a new matrix next to the max values from other patches.
- Fully connected layer: After stacking up a bunch of convolution and pooling layer, we follow them with a fully connected layer where we feed the extracted high-level features that we got from the input image to this fully connected layer to use them and do the actual classification based on these features. For our case, the fruit classification task, we follow the convolution and pooling layer with a fully connected layer that has 74 neurons and SoftMax activation to perform the actual classification. Finally, we use the dense () function of TensorFlow to define our fully connected layer with the required number of neurons and the final activation function.

So far, we have covered he technical details of how CNNs work and basic architecture of a CNN in TensorFlow. I will continue with Transfer learning Architecture which we used for our final solution for our fruit classification problem.

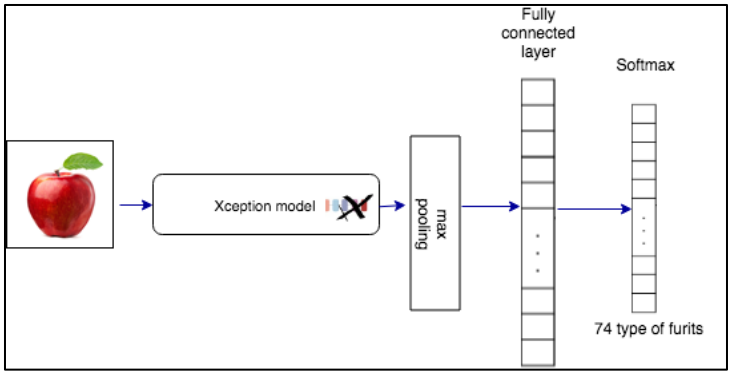
CNN architectures need a lot of data to perform well. Transfer learning

solves this problem by transferring gained knowledge from solving a task with a large dataset to another one with a smaller dataset. Training large deep learning architectures from scratch can sometimes be very slow because we have millions of weights in these architectures that need to be learned. Instead, we make use of Transfer learning by just fine-tuning a learned weight on our fruit classification problem.

In our case, we used a pre-trained convolution model on an ImageNet which is large dataset and adapt it to work on our problem.

we replaced the final fully connected layer of the pre-trained Xception model and then use the rest of the Xception model as a feature extractor. The last convolutional output of Xception is fed to the new model which has a global average pooling layer and a dense layer with SoftMax activation.

The following diagram, shows the general solution outline that we will be following:



Now that we have good understanding of CNN and transfer CNN architecture, I will explain the process of getting data and train my models.

After I downloaded data from Kaggle fruit data set, I removed images randomly from each category of fruits to reduce number of images.

Our image data split into three different sets

- Train set: This will be used as a knowledge base for our model. It has 6563 images of fruits
- Validation set: This will be used to choose the best performing model among a set of models. It has 2218 images of fruits
- Test set: This will be used to measure and report the accuracy of the selected model. It has 1985 images of fruits

I preprocessed and reshaped data to use Keras CNN library which require a 4D array as input.

To detect fruits in images, I used a pre-trained Xception model which have been trained on ImageNet which contains over 10 million images containing an object from one of 1000 categories.

I chose Xception model. I added a global average pooling layer and a dense layer with SoftMax activation. This model, which uses the pre-trained Xception model as a fixed feature extractor, where the last convolutional output of Xception is fed as input to our model. The model is trained for 20 epochs

I used Keras library to implement the convolutional neural network. I choose Xception because based on Keras documentation, it will accept 100 by 100 images while others had some restriction over width and height of image which did not work with my dataset. Also, it was one of the top accuracy rating according to Keras documentation. https://keras.io/applications/#xception

Benchmark

I used vanilla CNN and I created CNN from Scratch
The input to a convolutional layer is a m x m x r image where m is the height and width of the image and r is the number of channels, the convolutional layer will have k filters of size n x n x q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller. The first layer is a convolutional layer of shape 100 x 100 x 3. We choose 2 by 2 the length of the

convolution window and 16 number of output filters in the convolution for first layer.
we have Relu activation function to help us to alleviate the vanishing gradient problem
"same" padding will result in the output has the same length as the original input.
stride is one here, Stride controls how the filter convolves around the input volume
After each conventional layer, we apply a pooling layer, its job is to reduce the spatial size of the representation to reduce the number of parameters and computation in the network. We use max pooling which takes the largest value from one patch of an image, places it in a new matrix next to the max values from other patches.
we make each training pass over the data. Each node gets dropped off with a probability of 0.2. to avoid over fitting
For Fully Connected layer we use SoftMax activation function in the output layer. The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.
We used Keras library to implement the convolutional neural network. We used Conv2D, MaxPooling2D, Dense functions to implement above algorithm. We got our accuracy around 80.25%.
below is model summery

Layer Type	**Output Shape**	**Param #**
Convolutional	(None, 100, 100, 16)	208
Max Pooling	(None, 50, 50, 16)	0
Convolutional	(None, 50, 50, 32)	2080
Max Pooling	(None, 25, 25, 32)	0
Convolutional	(None, 25, 25, 64)	8256
Max Pooling	(None, 12, 12, 64)	0
Flatten	(None, 9216)	0
Dropout	(None, 9216)	0
Dense	(None, 74)	682058

```
=================================================================
Total params: 692,602
Trainable params: 692,602
Non-trainable params: 0
_________________________________________________________________
```

III. Methodology

Data Preprocessing

After I download fruit data from Kaggel. I removed some of data to make data size smaller. I used Keras, CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape (nb_samples, rows, columns, channels) where nb_samples correspond to the total number
of images, and rows, columns, and channels correspond to the number of rows, columns, and
channels for each image, respectively. For example, colored images have channel 3 as RGB. I first converted the data from image type to 3D tensor with shape (100, 100, 3) then I converted 3D tensor to 4D tensor with shape (1, 100, 100, 3).
below is code snippet for preprocessing the data

```python
from keras.preprocessing import image
from tqdm import tqdm

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(100, 100))
    # convert PIL.Image.Image type to 3D tensor with shape (100, 100, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 100, 100, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

Implementation

Loading and exploring

Let's start off by importing our dataset and have a look at the distribution of the training, validation and testing sets

```
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np
from glob import glob
# define function to load train, test, and validation datasets
def load_dataset(path):
    data = load_files(path)
    fruit_files = np.array(data['filenames'])
    fruit_targets = np_utils.to_categorical(np.array(data['target']), 74)
    return fruit_files, fruit_targets

# load train, test, and validation datasets
train_files, train_targets = load_dataset('fruitImages/Training')
valid_files, valid_targets = load_dataset('fruitImages/Validation')
test_files, test_targets = load_dataset('fruitImages/Test')

# load list of fruit names
fruit_names = [item[20:-1] for item in sorted(glob("fruitImages/Training/*
/"))]

# print statistics about the dataset
print('There are %d total fruit categories.' % len(fruit_names))
print('There are %s total fruit images.\n' % len(np.hstack([train_files, v
alid_files, test_files])))
print('There are %d training fruit images.' % len(train_files))
print('There are %d validation fruit images.' % len(valid_files))
print('There are %d test fruit images.'% len(test_files))
```

Output:

```
There are 74 total fruit categories.
There are 10766 total fruit images.

There are 6563 training fruit images.
There are 2218 validation fruit images.
There are 1985 test fruit images.
```

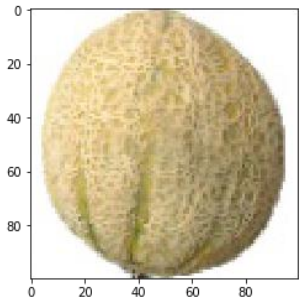
Let's define some helper functions that will enable us to explore the dataset. The following helper function plots a set of two images in a grid:

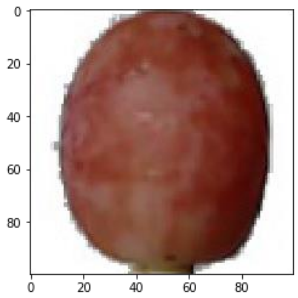
```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
```

```
fruit_files_sample_one = train_files[4]
fruit_files_sample_two = train_files[50]
def displayImage(img_path):
    img = cv2.imread(img_path)
    cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(cv_rgb)
    plt.show()

displayImage(fruit_files_sample_one)
displayImage(fruit_files_sample_two)
```

Output:





Let's preprocess data to convert it what TensorFlow needs

```
from keras.preprocessing import image
from tqdm import tqdm

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(100, 100))
    # convert PIL.Image.Image type to 3D tensor with shape (100, 100, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 100, 100, 3) and retur
n 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
```

```
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_p
aths)]
    return np.vstack(list_of_tensors)



from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

# pre-process the data for Keras
train_tensors = paths_to_tensor(train_files).astype('float32')/255
valid_tensors = paths_to_tensor(valid_files).astype('float32')/255
test_tensors = paths_to_tensor(test_files).astype('float32')/255
```

Xception model transfer values

As we mentioned earlier, we will be using the pre-trained Xception model on the ImageNet dataset. The last convolutional output of Xception is fed to the new model which has a global average pooling layer and a dense layer with SoftMax activation

```
from keras.applications.xception import Xception
from keras.models import Model
base_model=Xception(include_top=False, weights='imagenet', input_shape=(10
0, 100, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(74, activation='softmax')(x)
Xception_model = Model(base_model.input, predictions)
print(Xception_model.summary())
```

Output:

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_2 (InputLayer)         (None, 100, 100, 3)  0
__________________________________________________________________________________________________
block1_conv1 (Conv2D)        (None, 49, 49, 32)   864         input_2[0][0]
__________________________________________________________________________________________________
block1_conv1_bn (BatchNormaliza (None, 49, 49, 32)   128         block1_conv1[0][0]
__________________________________________________________________________________________________
block1_conv1_act (Activation)   (None, 49, 49, 32)   0           block1_conv1_bn[0][0]
__________________________________________________________________________________________________
block1_conv2 (Conv2D)        (None, 47, 47, 64)   18432       block1_conv1_act[0][0]
__________________________________________________________________________________________________
block1_conv2_bn (BatchNormaliza (None, 47, 47, 64)   256         block1_conv2[0][0]
__________________________________________________________________________________________________
block1_conv2_act (Activation)   (None, 47, 47, 64)   0           block1_conv2_bn[0][0]
__________________________________________________________________________________________________
block2_sepconv1 (SeparableConv2 (None, 47, 47, 128)  8768        block1_conv2_act[0][0]
__________________________________________________________________________________________________
block2_sepconv1_bn (BatchNormal (None, 47, 47, 128)  512         block2_sepconv1[0][0]
__________________________________________________________________________________________________
block2_sepconv2_act (Activation (None, 47, 47, 128)  0           block2_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block2_sepconv2 (SeparableConv2 (None, 47, 47, 128)  17536       block2_sepconv2_act[0][0]
```

block2_sepconv2_bn (BatchNormal (None, 47, 47, 128) 512 block2_sepconv2[0][0]

conv2d_8 (Conv2D) (None, 24, 24, 128) 8192 block1_conv2_act[0][0]

block2_pool (MaxPooling2D) (None, 24, 24, 128) 0 block2_sepconv2_bn[0][0]

batch_normalization_5 (BatchNor (None, 24, 24, 128) 512 conv2d_8[0][0]

add_13 (Add) (None, 24, 24, 128) 0 block2_pool[0][0]
batch_normalization_5[0][0]

block3_sepconv1_act (Activation (None, 24, 24, 128) 0 add_13[0][0]

block3_sepconv1 (SeparableConv2 (None, 24, 24, 256) 33920 block3_sepconv1_act[0][0]

block3_sepconv1_bn (BatchNormal (None, 24, 24, 256) 1024 block3_sepconv1[0][0]

block3_sepconv2_act (Activation (None, 24, 24, 256) 0 block3_sepconv1_bn[0][0]

block3_sepconv2 (SeparableConv2 (None, 24, 24, 256) 67840 block3_sepconv2_act[0][0]

block3_sepconv2_bn (BatchNormal (None, 24, 24, 256) 1024 block3_sepconv2[0][0]

conv2d_9 (Conv2D) (None, 12, 12, 256) 32768 add_13[0][0]

block3_pool (MaxPooling2D) (None, 12, 12, 256) 0 block3_sepconv2_bn[0][0]

batch_normalization_6 (BatchNor (None, 12, 12, 256) 1024 conv2d_9[0][0]

add_14 (Add) (None, 12, 12, 256) 0 block3_pool[0][0]
batch_normalization_6[0][0]

block4_sepconv1_act (Activation (None, 12, 12, 256) 0 add_14[0][0]

block4_sepconv1 (SeparableConv2 (None, 12, 12, 728) 188672 block4_sepconv1_act[0][0]

block4_sepconv1_bn (BatchNormal (None, 12, 12, 728) 2912 block4_sepconv1[0][0]

block4_sepconv2_act (Activation (None, 12, 12, 728) 0 block4_sepconv1_bn[0][0]

block4_sepconv2 (SeparableConv2 (None, 12, 12, 728) 536536 block4_sepconv2_act[0][0]

block4_sepconv2_bn (BatchNormal (None, 12, 12, 728) 2912 block4_sepconv2[0][0]

conv2d_10 (Conv2D) (None, 6, 6, 728) 186368 add_14[0][0]

block4_pool (MaxPooling2D) (None, 6, 6, 728) 0 block4_sepconv2_bn[0][0]

batch_normalization_7 (BatchNor (None, 6, 6, 728) 2912 conv2d_10[0][0]

add_15 (Add) (None, 6, 6, 728) 0 block4_pool[0][0]
batch_normalization_7[0][0]

block5_sepconv1_act (Activation (None, 6, 6, 728) 0 add_15[0][0]

block5_sepconv1 (SeparableConv2 (None, 6, 6, 728) 536536 block5_sepconv1_act[0][0]

block5_sepconv1_bn (BatchNormal (None, 6, 6, 728) 2912 block5_sepconv1[0][0]

block5_sepconv2_act (Activation (None, 6, 6, 728) 0 block5_sepconv1_bn[0][0]

block5_sepconv2 (SeparableConv2 (None, 6, 6, 728) 536536 block5_sepconv2_act[0][0]

block5_sepconv2_bn (BatchNormal (None, 6, 6, 728) 2912 block5_sepconv2[0][0]

block5_sepconv3_act (Activation (None, 6, 6, 728) 0 block5_sepconv2_bn[0][0]

block5_sepconv3 (SeparableConv2 (None, 6, 6, 728) 536536 block5_sepconv3_act[0][0]

block5_sepconv3_bn (BatchNormal (None, 6, 6, 728) 2912 block5_sepconv3[0][0]

add_16 (Add) (None, 6, 6, 728) 0 block5_sepconv3_bn[0][0]
add_15[0][0]

block6_sepconv1_act (Activation (None, 6, 6, 728) 0 add_16[0][0]

block6_sepconv1 (SeparableConv2 (None, 6, 6, 728) 536536 block6_sepconv1_act[0][0]

block6_sepconv1_bn (BatchNormal (None, 6, 6, 728) 2912 block6_sepconv1[0][0]

block6_sepconv2_act (Activation	(None, 6, 6, 728)	0	block6_sepconv1_bn[0][0]
block6_sepconv2 (SeparableConv2	(None, 6, 6, 728)	536536	block6_sepconv2_act[0][0]
block6_sepconv2_bn (BatchNormal	(None, 6, 6, 728)	2912	block6_sepconv2[0][0]
block6_sepconv3_act (Activation	(None, 6, 6, 728)	0	block6_sepconv2_bn[0][0]
block6_sepconv3 (SeparableConv2	(None, 6, 6, 728)	536536	block6_sepconv3_act[0][0]
block6_sepconv3_bn (BatchNormal	(None, 6, 6, 728)	2912	block6_sepconv3[0][0]
add_17 (Add)	(None, 6, 6, 728)	0	block6_sepconv3_bn[0][0] add_16[0][0]
block7_sepconv1_act (Activation	(None, 6, 6, 728)	0	add_17[0][0]
block7_sepconv1 (SeparableConv2	(None, 6, 6, 728)	536536	block7_sepconv1_act[0][0]
block7_sepconv1_bn (BatchNormal	(None, 6, 6, 728)	2912	block7_sepconv1[0][0]
block7_sepconv2_act (Activation	(None, 6, 6, 728)	0	block7_sepconv1_bn[0][0]
block7_sepconv2 (SeparableConv2	(None, 6, 6, 728)	536536	block7_sepconv2_act[0][0]
block7_sepconv2_bn (BatchNormal	(None, 6, 6, 728)	2912	block7_sepconv2[0][0]
block7_sepconv3_act (Activation	(None, 6, 6, 728)	0	block7_sepconv2_bn[0][0]
block7_sepconv3 (SeparableConv2	(None, 6, 6, 728)	536536	block7_sepconv3_act[0][0]
block7_sepconv3_bn (BatchNormal	(None, 6, 6, 728)	2912	block7_sepconv3[0][0]
add_18 (Add)	(None, 6, 6, 728)	0	block7_sepconv3_bn[0][0] add_17[0][0]
block8_sepconv1_act (Activation	(None, 6, 6, 728)	0	add_18[0][0]
block8_sepconv1 (SeparableConv2	(None, 6, 6, 728)	536536	block8_sepconv1_act[0][0]
block8_sepconv1_bn (BatchNormal	(None, 6, 6, 728)	2912	block8_sepconv1[0][0]
block8_sepconv2_act (Activation	(None, 6, 6, 728)	0	block8_sepconv1_bn[0][0]
block8_sepconv2 (SeparableConv2	(None, 6, 6, 728)	536536	block8_sepconv2_act[0][0]
block8_sepconv2_bn (BatchNormal	(None, 6, 6, 728)	2912	block8_sepconv2[0][0]
block8_sepconv3_act (Activation	(None, 6, 6, 728)	0	block8_sepconv2_bn[0][0]
block8_sepconv3 (SeparableConv2	(None, 6, 6, 728)	536536	block8_sepconv3_act[0][0]
block8_sepconv3_bn (BatchNormal	(None, 6, 6, 728)	2912	block8_sepconv3[0][0]
add_19 (Add)	(None, 6, 6, 728)	0	block8_sepconv3_bn[0][0] add_18[0][0]
block9_sepconv1_act (Activation	(None, 6, 6, 728)	0	add_19[0][0]
block9_sepconv1 (SeparableConv2	(None, 6, 6, 728)	536536	block9_sepconv1_act[0][0]
block9_sepconv1_bn (BatchNormal	(None, 6, 6, 728)	2912	block9_sepconv1[0][0]
block9_sepconv2_act (Activation	(None, 6, 6, 728)	0	block9_sepconv1_bn[0][0]
block9_sepconv2 (SeparableConv2	(None, 6, 6, 728)	536536	block9_sepconv2_act[0][0]
block9_sepconv2_bn (BatchNormal	(None, 6, 6, 728)	2912	block9_sepconv2[0][0]
block9_sepconv3_act (Activation	(None, 6, 6, 728)	0	block9_sepconv2_bn[0][0]
block9_sepconv3 (SeparableConv2	(None, 6, 6, 728)	536536	block9_sepconv3_act[0][0]
block9_sepconv3_bn (BatchNormal	(None, 6, 6, 728)	2912	block9_sepconv3[0][0]
add_20 (Add)	(None, 6, 6, 728)	0	block9_sepconv3_bn[0][0] add_19[0][0]
block10_sepconv1_act (Activatio	(None, 6, 6, 728)	0	add_20[0][0]

block10_sepconv1 (SeparableConv	(None, 6, 6, 728)	536536	block10_sepconv1_act[0][0]
block10_sepconv1_bn (BatchNorma	(None, 6, 6, 728)	2912	block10_sepconv1[0][0]
block10_sepconv2_act (Activatio	(None, 6, 6, 728)	0	block10_sepconv1_bn[0][0]
block10_sepconv2 (SeparableConv	(None, 6, 6, 728)	536536	block10_sepconv2_act[0][0]
block10_sepconv2_bn (BatchNorma	(None, 6, 6, 728)	2912	block10_sepconv2[0][0]
block10_sepconv3_act (Activatio	(None, 6, 6, 728)	0	block10_sepconv2_bn[0][0]
block10_sepconv3 (SeparableConv	(None, 6, 6, 728)	536536	block10_sepconv3_act[0][0]
block10_sepconv3_bn (BatchNorma	(None, 6, 6, 728)	2912	block10_sepconv3[0][0]
add_21 (Add)	(None, 6, 6, 728)	0	block10_sepconv3_bn[0][0] add_20[0][0]
block11_sepconv1_act (Activatio	(None, 6, 6, 728)	0	add_21[0][0]
block11_sepconv1 (SeparableConv	(None, 6, 6, 728)	536536	block11_sepconv1_act[0][0]
block11_sepconv1_bn (BatchNorma	(None, 6, 6, 728)	2912	block11_sepconv1[0][0]
block11_sepconv2_act (Activatio	(None, 6, 6, 728)	0	block11_sepconv1_bn[0][0]
block11_sepconv2 (SeparableConv	(None, 6, 6, 728)	536536	block11_sepconv2_act[0][0]
block11_sepconv2_bn (BatchNorma	(None, 6, 6, 728)	2912	block11_sepconv2[0][0]
block11_sepconv3_act (Activatio	(None, 6, 6, 728)	0	block11_sepconv2_bn[0][0]
block11_sepconv3 (SeparableConv	(None, 6, 6, 728)	536536	block11_sepconv3_act[0][0]
block11_sepconv3_bn (BatchNorma	(None, 6, 6, 728)	2912	block11_sepconv3[0][0]
add_22 (Add)	(None, 6, 6, 728)	0	block11_sepconv3_bn[0][0] add_21[0][0]
block12_sepconv1_act (Activatio	(None, 6, 6, 728)	0	add_22[0][0]
block12_sepconv1 (SeparableConv	(None, 6, 6, 728)	536536	block12_sepconv1_act[0][0]
block12_sepconv1_bn (BatchNorma	(None, 6, 6, 728)	2912	block12_sepconv1[0][0]
block12_sepconv2_act (Activatio	(None, 6, 6, 728)	0	block12_sepconv1_bn[0][0]
block12_sepconv2 (SeparableConv	(None, 6, 6, 728)	536536	block12_sepconv2_act[0][0]
block12_sepconv2_bn (BatchNorma	(None, 6, 6, 728)	2912	block12_sepconv2[0][0]
block12_sepconv3_act (Activatio	(None, 6, 6, 728)	0	block12_sepconv2_bn[0][0]
block12_sepconv3 (SeparableConv	(None, 6, 6, 728)	536536	block12_sepconv3_act[0][0]
block12_sepconv3_bn (BatchNorma	(None, 6, 6, 728)	2912	block12_sepconv3[0][0]
add_23 (Add)	(None, 6, 6, 728)	0	block12_sepconv3_bn[0][0] add_22[0][0]
block13_sepconv1_act (Activatio	(None, 6, 6, 728)	0	add_23[0][0]
block13_sepconv1 (SeparableConv	(None, 6, 6, 728)	536536	block13_sepconv1_act[0][0]
block13_sepconv1_bn (BatchNorma	(None, 6, 6, 728)	2912	block13_sepconv1[0][0]
block13_sepconv2_act (Activatio	(None, 6, 6, 728)	0	block13_sepconv1_bn[0][0]
block13_sepconv2 (SeparableConv	(None, 6, 6, 1024)	752024	block13_sepconv2_act[0][0]
block13_sepconv2_bn (BatchNorma	(None, 6, 6, 1024)	4096	block13_sepconv2[0][0]
conv2d_11 (Conv2D)	(None, 3, 3, 1024)	745472	add_23[0][0]
block13_pool (MaxPooling2D)	(None, 3, 3, 1024)	0	block13_sepconv2_bn[0][0]
batch_normalization_8 (BatchNor	(None, 3, 3, 1024)	4096	conv2d_11[0][0]

```
add_24 (Add)                 (None, 3, 3, 1024)   0          block13_pool[0][0]
                                                             batch_normalization_8[0][0]
__________________________________________________________________________________________________
block14_sepconv1 (SeparableConv (None, 3, 3, 1536)   1582080    add_24[0][0]
__________________________________________________________________________________________________
block14_sepconv1_bn (BatchNorma (None, 3, 3, 1536)   6144       block14_sepconv1[0][0]
__________________________________________________________________________________________________
block14_sepconv1_act (Activatio (None, 3, 3, 1536)   0          block14_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block14_sepconv2 (SeparableConv (None, 3, 3, 2048)   3159552    block14_sepconv1_act[0][0]
__________________________________________________________________________________________________
block14_sepconv2_bn (BatchNorma (None, 3, 3, 2048)   8192       block14_sepconv2[0][0]
__________________________________________________________________________________________________
block14_sepconv2_act (Activatio (None, 3, 3, 2048)   0          block14_sepconv2_bn[0][0]
__________________________________________________________________________________________________
global_average_pooling2d_1 (Glo (None, 2048)         0          block14_sepconv2_act[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)              (None, 74)            151626     global_average_pooling2d_1[0][0]
==================================================================================================
Total params: 21,013,106
Trainable params: 20,958,578
Non-trainable params: 54,528
__________________________________________________________________________________________________
None
```

Train the Model

For the optimizer in the CNN model, I used two different optimizers with my model and in the below chart the accuracy of model is shown

Optimizer	Accuracy
nadam	90.47%
rmsprop	89.77%

Finally, Nadam was used as an optimizer because it has better accuracy compare to RMSprop. Nadam is like Adam is essentially RMSprop with momentum, Nadam is Adam RMSprop with Nesterov momentum.
https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Nadam.
Loss parameter is categorical_crossentropy https://keras.io/losses/
Keras allows you to specifying the 'metrics' argument to monitor during the training of your model. Metric values are recorded at the end of each epoch on the training dataset. I set Metric as accuracy and epochs to 5.

```
Xception_model.compile(loss='categorical_crossentropy', optimizer='nadam',
 metrics=['accuracy'])

### train the model
checkpointer2 = ModelCheckpoint(filepath='saved_models/weights.best.Xcepti
on.hdf5',
                                verbose=1, save_best_only=True)

Xception_model.fit(train_tensors, train_targets,
          validation_data=(valid_tensors, valid_targets),
```

```
          epochs=5, batch_size=200, callbacks=[checkpointer2], verbose=1)

### load the model with the best Validation Loss
Xception_model.load_weights('saved_models/weights.best.Xception.hdf5')
```

Refinement

I adjusted batch size parameters to decrease training time because of my machine hardware
Also, I changed epochs number to see if I get better numbers
I decreased training size, since my machine could not handle so many images. So, I reduced training data set size to 6563 number of fruit images.
I changed optimizer from RMSprop to Nadam because I get better result after testing my model.

Optimizer	Accuracy
nadam	90.47%
rmsprop	89.77%

IV. Results

Model Evaluation and Validation

The final model uses transfer learning to create a CNN using Xception pre-computed
features that are currently available in Keras.
The last convolutional output of Xception is fed as input to my model. I will add a global average pooling layer and a dense layer with Softmax activation.

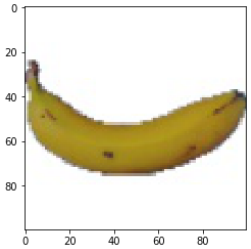
```
1. from keras.applications.xception import Xception
2. base_model=Xception(include_top=False, weights='imagenet', input_shape=(100, 100, 3))
3. x = base_model.output
4. x = GlobalAveragePooling2D()(x)
5. predictions = Dense(74, activation='softmax')(x)
6. Xception_model = Model(base_model.input, predictions)
7. print(Xception_model.summary())
```

Total params: 21,013,106
Trainable params: 20,958,578
Non-trainable params: 54,528

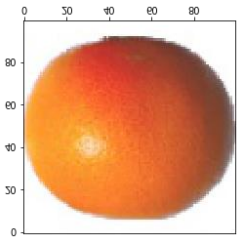
There will be 74 classifiers at end.
Loss parameter is categorical_crossentropy.
Metric values are recorded at the end of each epoch on the training dataset. I set Metric as accuracy.
For the optimizer in the CNN model, I used two different optimizers with my model and in the below chart the accuracy of model is shown

Optimizer	Accuracy
nadam	90.47%
rmsprop	89.77%

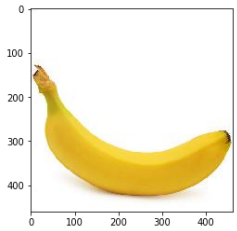
My model did not respond well to some of unseen data. I downloaded 7 random images from Google and here is the result.



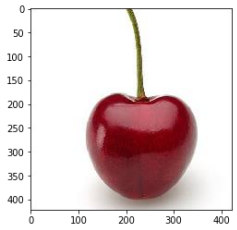
```
predicted fruit type is: \Banana
Actual: Banana
```



```
predicted fruit type is: \Grapefruit Pink
Actual: Grapefruit Pink
```

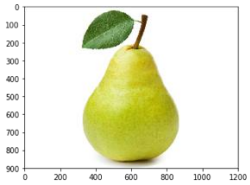


predicted fruit type is: \Banana Red
Actual: Banana

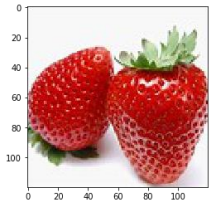


predicted fruit type is: \Cactus fruit
Actual: Cherry

predicted fruit type is: \Physalis
Actual: Grapes



predicted fruit type is: \Pear
Actual: Pear



predicted fruit type is: \Strawberry
Actual: Strawberry

Justification

The below chart shows the difference between benchmark and transfer modeling. The result from CNN transfer model is better than vanilla CNN from scratch

	training image	accuracy	optimizer	Epoch
Vanilla CNN	6563	80.25%	rmsprop	5
CNN with transfer	6563	89.77%	rmsprop	5
CNN with transfer	6563	90.47%	nadam	5

V. Conclusion

Free-Form Visualization:

I represent confusion matrix which is used to evaluate the quality of the output of a classifier on the fruit data set. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions. [3]

Below table show number of test images for each fruit label in my test folder

	number of images in Testing
Apple Braeburn	7.0
Apple Golden 1	7.0
Apple Golden 2	7.0
Apple Golden 3	7.0
Apple Granny Smith	7.0
Apple Red 1	7.0
Apple Red 2	7.0
Apple Red 3	6.0
Apple Red Delicious	41.0
Apple Red Yellow	7.0
Apricot	7.0
Avocado ripe	6.0
Avocado	65.0
Banana Red	34.0
Banana	41.0
Cactus fruit	13.0
Cantaloupe 1	59.0
Cantaloupe 2	43.0
Carambula	31.0
Cherry 1	7.0
Cherry 2	91.0
Cherry Rainier	127.0
Cherry Wax Black	31.0
Cherry Wax Red	24.0
Cherry Wax Yellow	14.0
Clementine	35.0
Cocos	16.0
Dates	50.0
Granadilla	23.0
Grape Pink	7.0
Grape White 2	4.0
Grape White	15.0
Grapefruit Pink	13.0
Grapefruit White	7.0
Guava	17.0
Huckleberry	28.0
Kaki	37.0
Kiwi	7.0
Kumquats	37.0
Lemon Meyer	7.0
Lemon	35.0
Limes	62.0
Lychee	11.0
Mandarine	16.0
Mango	48.0
Maracuja	66.0
Melon Piel de Sapo	93.0
Mulberry	27.0
Nectarine	7.0
Orange	7.0
Papaya	7.0
Passion Fruit	66.0
Peach Flat	7.0
Peach	7.0
Pear Abate	7.0
Pear Monster	6.0
Pear Williams	26.0
Pear	6.0
Pepino	8.0
Physalis with Husk	21.0
Physalis	31.0
Pineapple Mini	24.0
Pineapple	24.0
Pitahaya Red	22.0
Plum	7.0
Pomegranate	7.0
Quince	40.0
Rambutan	66.0
Raspberry	14.0
Salak	48.0
Strawberry Wedge	7.0
Strawberry	136.0
Tamarillo	4.0
Tangelo	23.0

This is how to read and interpret below matrices:
The x axis is predicted result and y axis is actual label. The expected output should be the main diagonal of the matrix with a 1.0 score in normalized figure [1]; that is, all the expected values should match the real ones.
If you zoom in Figure [2] , you will see number in each cell in the main diagonal .
The bigger the numbers on the matrix diagonally, the better.

- For Apple Braeburn, we have 7 images in the test dataset. Now if you zoom in at the in confusions matrix[2], you will see 7 of them predicted correctly
- For Strawberry, we have 136 images, Now if you zoom in at the in confusions matrix[2], you will see 80 of them predicted correctly. 34 of them predicted as Salak , 10 predicted Rambutan and 12 predicted wrongly as date.

Salak

Rambutan

date

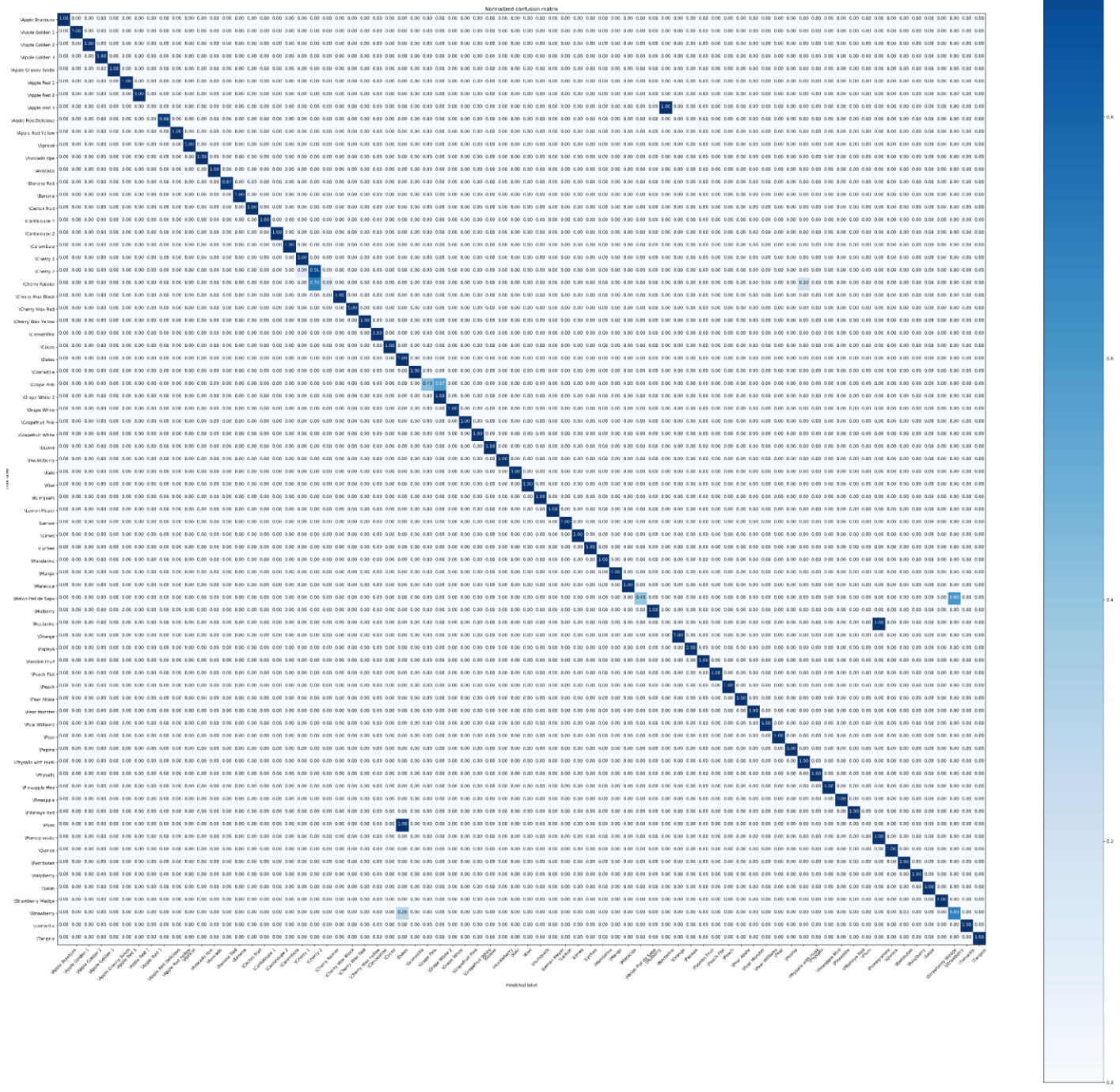


Figure 1: Confusion matrix with normalization. Please zoom in to see the labels of the figure.

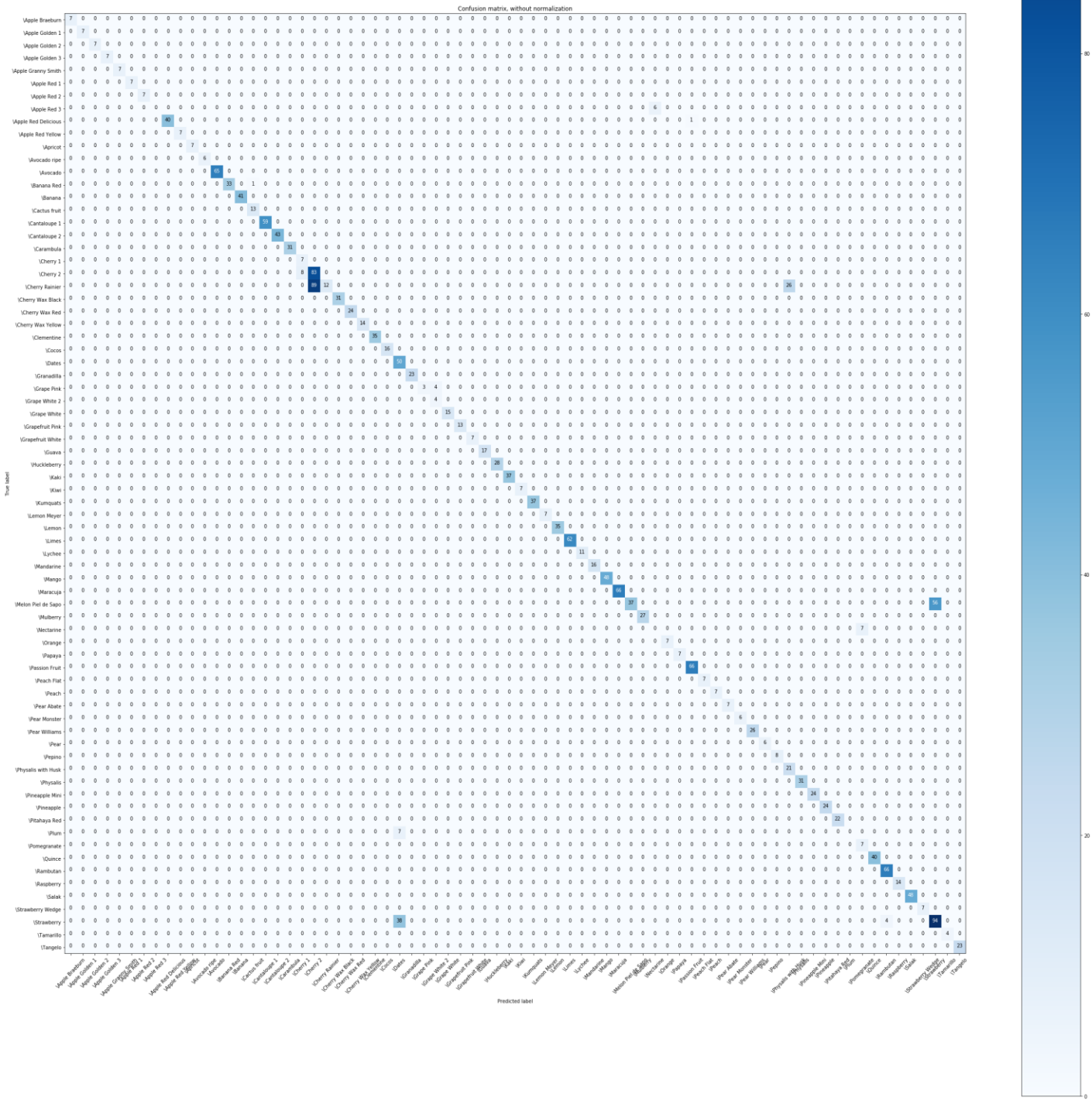


Figure 2: Confusion without normalization. Please zoom in to see the labels of the figure

Reflection

We investigate a problem of fruits classification. Given an image, we want to find if the image
contains fruit or not and what type of fruit is it. I achieved this by using Convolutional Neural Networks which are hierarchical neural networks for image classification.

First, I loaded fruit images data set for training and validation purpose.
Then I reduced number of images in each category of fruit then I preprocessed all images to feed to tensor flow which uses Keras CNNs library which require a 4D array as input, with
shape (nb_samples, rows, columns, channels) where nb_samples correspond to the total number
of images, and rows, columns, and channels correspond to the number of rows, columns, and
channels for each image, respectively.

Benchmark model: I just used vanilla CNN with three layers.

My final model used transfer learning to create a CNN using Xception pre-computed
features that are currently available in Keras.
The last convolutional output of Xception is fed as input to my model. I added a global average pooling layer and a dense layer with Softmax activation.

At the end of this project, I will accept any user-supplied image as input. If a fruit is detected in
the image, then it will provide an estimate of the type of fruits.

I learnt about process of creating transfer model an also how to create model from scratch using Keras. I found out my model respond well to images which has white

background. My model did not respond very well to unseen data, for future improvement, I can use more convolution layer after transfer model

Improvement

For improvement, different model architectures can be explored to reduce prediction time while maintaining accuracy or add more convolution and pooling layers after loading the pre-trained Xception model or change parameters of fully connected layers.
Also, providing nutrition facts about each fruit by having a map from fruit label to a sentence which includes nutrition facts about fruit can be a further improvement for this project.

Below describe common strategies for improving the performance

- Adding more layers
- Increasing the number of training epochs
- Experimenting with different activation functions

Reference

[1] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez & C. McCool, DeepFruits: A Fruit Detection System Using Deep Neural Networks, Sensors (Basel, Switzerland), Vol. 16(8), pp. 1222-, 2016.
[2] Fruits 360 Dataset on Kaggle. https://www.kaggle.com/moltean/
[3] http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html