Capstone 2: Image Recognition Fruit Classifier

- 5 classes [Red Apples, Blackberry, Green Grapes, Kiwi, Strawberry]
- CNN
- Google Colab

Rishi Phule

Goal: Recognize 5 Fruits with > 90% accuracy on unseen data.

Challenges: Learn the different variations of

1. Colors – Red Apples and Strawberries, Green Grapes and Kiwi
2. Shapes – Irregular shapes of Apples, Strawberries and Grapes
3. Partial Object recognition

Steps:

1. Data collection and curation
2. Data cleaning
3. Data wrangling – conversion and preprocessing
4. Training basic model
5. Data Augmentation
6. Model Training and Testing
7. Advanced model tuning
8. Conclusion

Out of Scope:

1. Sliced versions of fruits, except Kiwi
2. Recognize stage of fruit from raw to ripe

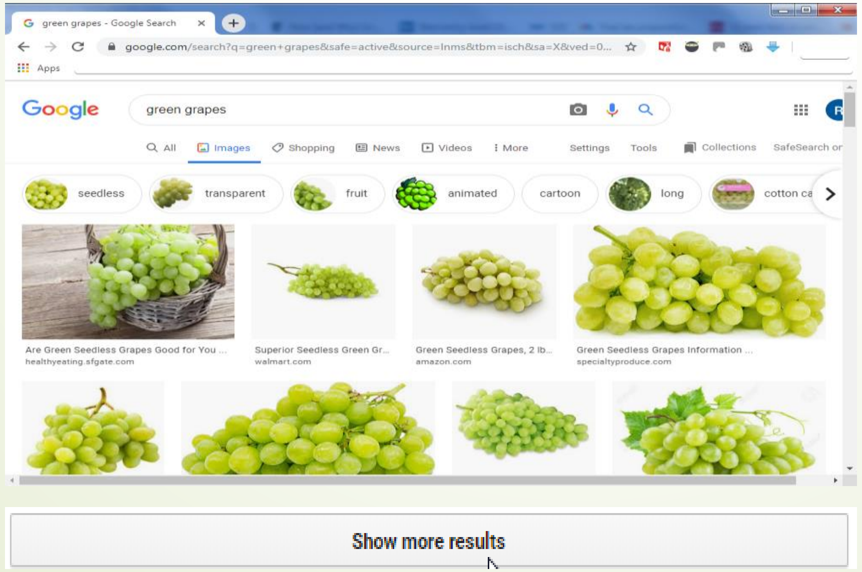
1. **Data Collection and Curation**

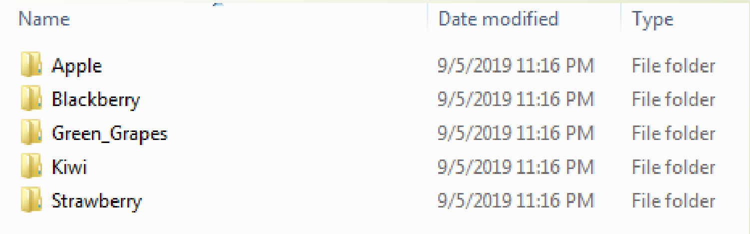
A. Flickr API

- API Registration with Key + Secret to be used in Python function
- 1000 images/ fruit downloaded with ‘relevance’
- Size = square (150x150), bigger than target size = (100x100)

B. Google Image Search

- Search keyword and scroll till end
- Run JavaScript to download URLs.txt of all images in window
- Run python function to download images for all links in URLs.txt





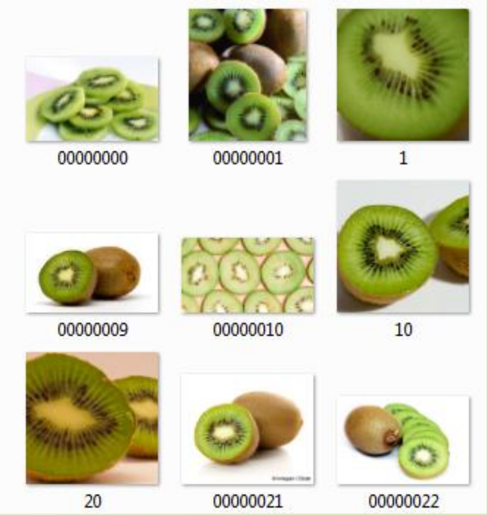
2. Data cleaning

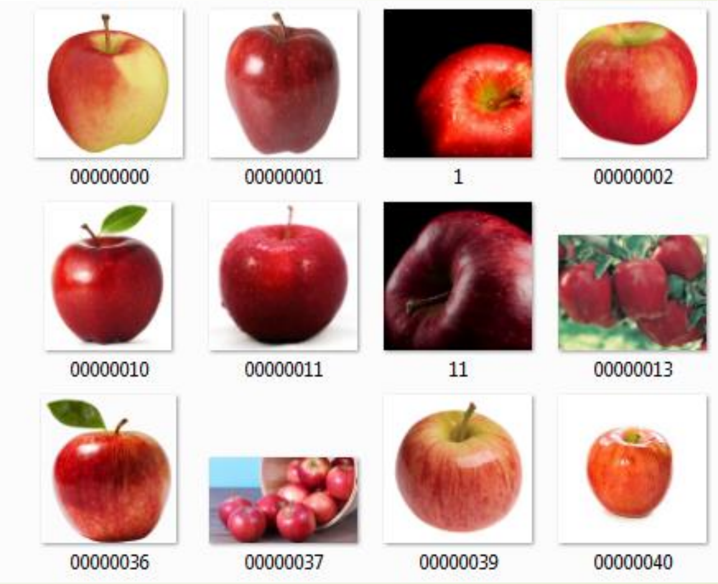
A. Auto-deletion via OpenCV

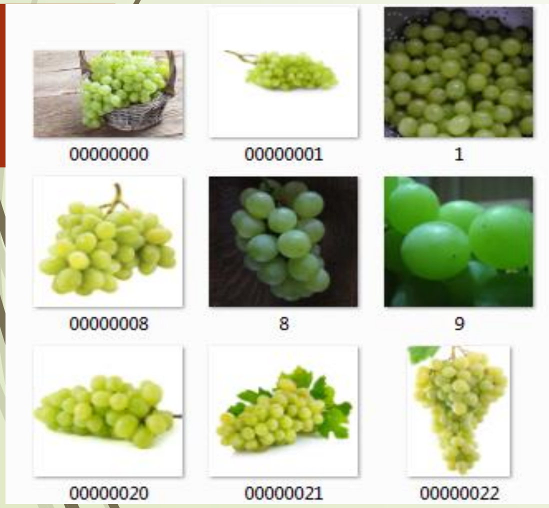
- Bad images – blanks, cannot be found etc.
- If file cannot be loaded, delete after downloading

B. Manual culling

- Delete if:
  - ❖ Watermarked, paintings, 3D Studio etc.
  - ❖ Irrelevant, not whole objects (Kiwi exception)
  - ❖ Presence of other fruits
  - ❖ Distance of object
  - ❖ Too small, too big



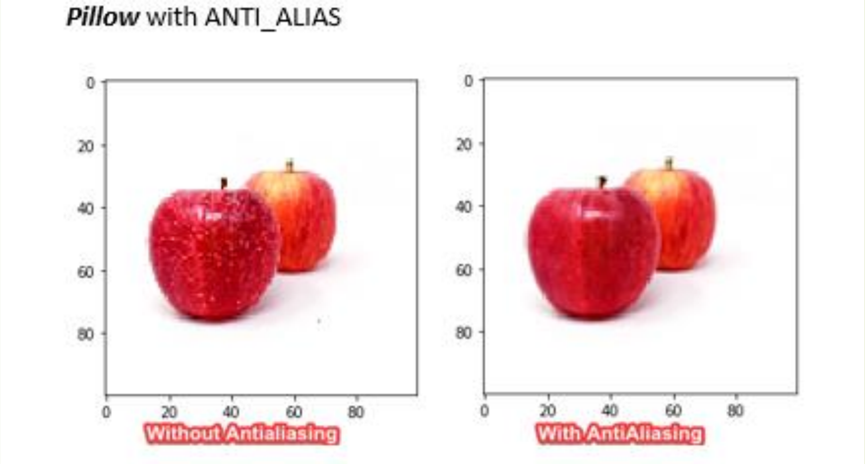


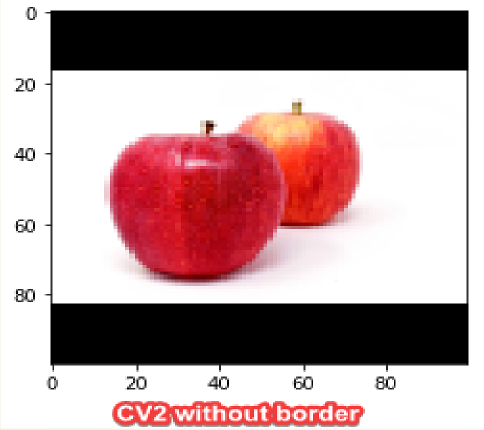


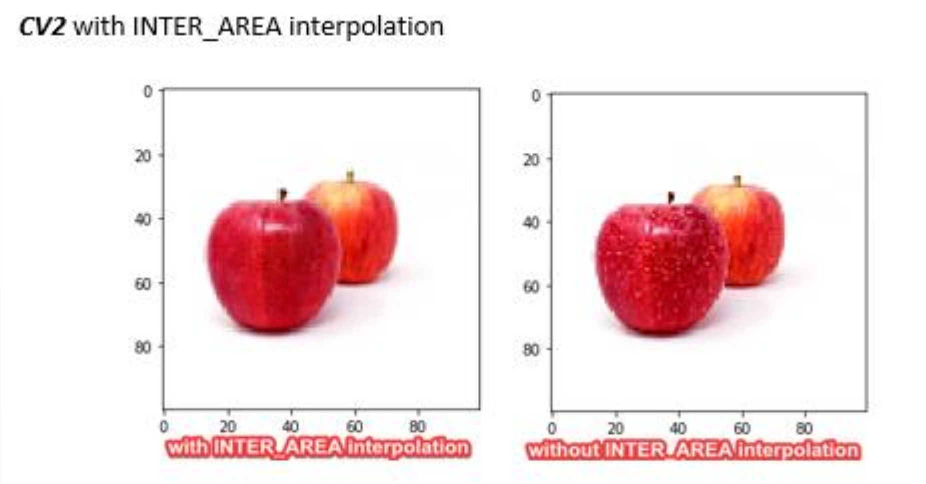
3. Data wrangling – conversion and preprocessing

Goal: Get best quality 100x100 images
Libraries: Pillow vs CV2

	Pillow	**CV2**
Display	imshow()	numpy array
Resizing	AntiAlias method	InterArea interpolation
Image weight	~ 2-3k	~4-6k
Padding	Blank image paste	Border coloring
Performance (GrayScale) Basic Classifier 10 Epochs	39s 80ms/step - loss: 0.1875 - acc: 0.9125 - val_loss: 0.1700 - val_acc: 0.9333	36s 76ms/step - loss: 0.1413 - acc: 0.9458 - val_loss: 0.0857 - val_acc: 0.9667







4. Train Basic Model

```
model3 = Sequential()
model3.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(100,100,3), padding='same'))
model3.add(Conv2D(32, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))
model3.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))
model3.add(Flatten())
model3.add(Dense(512, activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(5, activation='softmax'))
model3.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model3.fit(X_train, y_train, epochs=60, validation_data=(X_test, y_test), verbose=2)
```

```
 - 67s - loss: 0.0129 - acc: 0.9950 - val_loss: 1.0358 - val_acc: 0.8000
Epoch 53/60
 - 67s - loss: 0.0104 - acc: 0.9958 - val_loss: 1.0623 - val_acc: 0.8233
Epoch 54/60
 - 67s - loss: 0.0097 - acc: 0.9958 - val_loss: 1.2012 - val_acc: 0.8100
Epoch 55/60
 - 67s - loss: 0.0096 - acc: 0.9958 - val_loss: 1.0871 - val_acc: 0.8267
Epoch 56/60
 - 67s - loss: 0.0254 - acc: 0.9967 - val_loss: 1.5769 - val_acc: 0.7700
Epoch 57/60
 - 67s - loss: 0.0594 - acc: 0.9792 - val_loss: 1.1930 - val_acc: 0.8233
Epoch 58/60
 - 67s - loss: 0.0282 - acc: 0.9917 - val_loss: 1.0721 - val_acc: 0.8000
Epoch 59/60
 - 67s - loss: 0.0206 - acc: 0.9942 - val_loss: 1.0258 - val_acc: 0.8133
Epoch 60/60
 - 67s - loss: 0.0150 - acc: 0.9967 - val_loss: 1.0457 - val_acc: 0.8000
Wall time: 1h 7min 7s

<keras.callbacks.History at 0x48318d08>
```

- **Parameters:**
  - 4 Convolutional layers of kernel_size (3x3)
  - MaxPooling(2x2), Dropout = 0.25/0.5
  - Adam optimizer, loss=categorical_crossentropy, Time taken > 1hr
- **Learnings:**
  - Expensive to train locally
  - Overfitting on training set

Enter Google Colab

- **Specs per project and requirement:**
  - 12 GB GPU RAM
  - 350 GB of disk
  - Option to run it on GPU / Tensor / None
  - Data set uploaded to Google Drive and needs authentication
  - Fast: 5s / epoch vs. 67s / epoch

```
# Initiate Python in Colab environment
!pip install PyDrive
```

```
Collecting PyDrive
  Downloading https://files.pythonhosted.org/packages/52/e0/0e64788e5dd58ce2d6934549676243dc69d982f198524be9b99e9c2a4f
    |████████████████████████████████| 993kB 2.8MB/s
Requirement already satisfied: google-api-python-client>=1.2 in /usr/local/lib/python3.6/dist-packages (from PyDrive)
Requirement already satisfied: oauth2client>=4.0.0 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (4.1.3)
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (3.13)
Requirement already satisfied: httplib2<1dev,>=0.9.2 in /usr/local/lib/python3.6/dist-packages (from google-api-python
Requirement alr
Requirement alr
Requirement alr
Requirement alr
Requirement alr
Requirement alr
Requirement alr
Requirement alr
Building wheels
  Building whee
  Created wheel
  Stored in dir
Successfully bu
Installing coll
Successfully in
```

```
# Import essentials

import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
# Enable Google drive authentication and authorization to read from the google drive

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
# Link the training set, download and unzip it
download = drive.CreateFile({'id': '1gvdIbq3G4JwHvRUGvW7KnWA9hP2bpUSy'})
download.GetContentFile('Full_Set_Proc_Comb.zip')
!unzip Full_Set_Proc_Comb.zip
```

```
# Link the training set
download = drive.CreateFile({'id': '1mM5e0SRKuTklmVoKtqKx8dT_mxezrhxo'})
download.GetContentFile('Test_Set_Proc_Comb.zip')
!unzip Test_Set_Proc_Comb.zip
```

```
# Link the training set
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical
from keras.preprocessing import image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from tqdm import tqdm
```

```
Using TensorFlow backend.
```

```
# Read the train and test files
train = pd.read_csv('Full_Set_Proc_Comb/train.csv')
test = pd.read_csv('Test_Set_Proc_Comb/test.csv')
```

```
# Get full file names
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
df = train
df['id'] = df.id.astype(str)
df['label'] = df.label.astype(str)
df['id'] = df['id'] + '.jpg'
train_set, test_set = train_test_split(df, test_size=0.2, random_state=42)
train_set.head()
```

	id	label
382	383.jpg	1
538	539.jpg	1
1493	1494.jpg	4
1112	1113.jpg	3

5. Data Augmentation – using ImageDataGenerator of Keras

Augmentation Parameters:

- rotation_range = 40
- width_shift_range = 0.2
- height_shift_range = 0.2
- Rescale – 1./255
- shear_range = 0.2
- zoom_range = 0.2
- horizontal_flip = True

Three way to make data flow:

1. flow
2. flow_from_directory
3. flow_from_dataframe

Different methods with positives and negatives

flow	➢ Input = data & label arrays, output = batches of augmented data ➢ Needs images to be loaded, converted to arrays, reshaped ➢ -ve: Additional processing is needed for the steps of loading the images
flow_from_directory	➢ Reads images directly from directory and it takes labels as the directory names Images need to be organized in folders ➢ Parameter 'validation_split' ➢ -ve: train-test split was very uneven and sometimes has produced very high class imbalance.
flow_from_dataframe	➢ Takes pandas DataFrame containing names of files and classes as input ➢ Absolute and relative paths ➢ Flexible and use of sklearn.model_selection train_test_split

6. Model training and testing

Both training parameters (loss and accuracy) and validation parameters (val_loss and val_acc) were taken in to account during experimenting with different ideas

	Setting Tried	Summary
1	'vertical_flip' = True	Stalled the model's accuracy rate at around 86%
2	Increasing number of filters	Adjusting filters in layers of (32, 32, 64, 64) to 128 etc. caused increase in training time with no benefit to accuracy
3	Increasing convolutional layers	Adding additional layers increased run per epoch time, without any accuracy benefit. Leanear = better
4	Changing dropouts	Increasing reduced accuracy, decreasing introduced overfitting
5	Changing kernel_size	Increasing caused loss of accuracy, decreasing caused increase in training time and sensitivity
6	Changing Dense layer size	Increasing caused suboptimal performance as parameters increased exponentially. Back propogating also taking longer.

Conclusion:

- Sophistication / Complexity of a model comes at a computational and timing cost and may act in detriment of a simple classification neural network.
- If images were larger or complex, a more deeper and denser CNN would have been required.

7. Advanced tuning and testing

1. Checkpoint callback:

- During evaluation of model, the parameters from last epoch play significant role.
- Timing issue can arise and result in undermining accuracy

Example: Running validation after running 250 epochs

Validation set evaluation = [0.8309193852904718, 0.9]

Test set evaluation = [0.7641155040264129, 0.8599999904632568]

Callback feature saves model with best val_loss and continuously updates it

Validation on best performing weights:

Validation set evaluation = [0.2686952355752389, 0.97]

Test set evaluation = [0.2596906507015228, 0.9000000023841858]

2. Batch normalization:

- All convolutional and Dense layers get extra step of BatchNormlization before activation function.
- Needs ‘use_bias’=False in these layers
- BatchNormalization results in mean around 0 and standard deviation below 1 for 1 batch.
- Applied on axis=3 due to Keras running on TensorFlow with input tensor of [b, h, w, c] (batch_size, height, width and channels).

Model

```python
from keras import layers
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),input_shape=(100,100,3), padding='same', use_bias
= 'False'))
model.add(layers.BatchNormalization(axis=3))
model.add(layers.Activation('relu'))
model.add(Conv2D(32, (3, 3), use_bias = 'False'))
model.add(layers.BatchNormalization(axis=3))
model.add(layers.Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), use_bias = 'False', padding='same'))
model.add(layers.BatchNormalization(axis=3))
model.add(layers.Activation('relu'))
model.add(Conv2D(64, (3, 3), use_bias = 'False'))
model.add(layers.BatchNormalization(axis=3))
model.add(layers.Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, use_bias = 'False'))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(5, use_bias = 'False'))
model.add(layers.BatchNormalization())
model.add(layers.Activation('softmax'))
```

Model Summary

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 100, 100, 32)      896
_________________________________________________________________
batch_normalization_1 (Batch (None, 100, 100, 32)      128
_________________________________________________________________
activation_1 (Activation)    (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 98, 98, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 98, 98, 32)        128
_________________________________________________________________
activation_2 (Activation)    (None, 98, 98, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 49, 49, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 49, 49, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 49, 49, 64)        256
_________________________________________________________________
activation_3 (Activation)    (None, 49, 49, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 47, 47, 64)        36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 47, 47, 64)        256
_________________________________________________________________
activation_4 (Activation)    (None, 47, 47, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 23, 23, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 33856)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               4333696
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512
_________________________________________________________________
activation_5 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645
_________________________________________________________________
batch_normalization_6 (Batch (None, 5)                 20
_________________________________________________________________
activation_6 (Activation)    (None, 5)                 0
=================================================================
Total params: 4,401,209
Trainable params: 4,400,559
Non-trainable params: 650
```

Results

Validation set evaluation = [0.2686952355752389, 0.97]
Test set evaluation = [0.2596906507015228, 0.9000000023841858]

```
Confusion Matrix
[[ 8  0  1  0  1]
 [ 0 10  0  0  0]
 [ 0  1  9  0  0]
 [ 0  0  0 10  0]
 [ 0  0  0  0 10]]
```

8. Conclusion

Classification Report

	precision	recall	f1-score	support
Apple	1.00	0.80	0.89	10
Blackberry	0.91	1.00	0.95	10
Green Grapes	0.90	0.90	0.90	10
Kiwi	1.00	1.00	1.00	10
Strawberry	0.91	1.00	0.95	10
accuracy			0.94	50
macro avg	0.94	0.94	0.94	50
weighted avg	0.94	0.94	0.94	50

- Apples misclassified as with Strawberries expected
- Unexpected :
  - Apples misclassified as Grapes
  - Grapes misclassified as Blackberries
- Lighting conditions, edge detection, color issues
- Meets all goals with 94% accuracy on unseen data

Next steps and further investigation

- What images compromise accuracy (lighting, colors, mixed ?)\
- Adjust other parameter of ImageDataGenerator for increasing training set
- Broaden testing (unseen set)
- Use Keras image transformation to skip extra step of preprocessing and try out other image augmentation libraries such as ImgAug
- Use better training set (see bad examples below)