Image Classifier for Fruits Dataset - Team 4 Submission for SA4108 CA

Part 2: Convolutional Neural Network

Model 1: Handpicked Validation Data, Adam and RMSprop optimization algorithms; 200 x 200 px

---

In [ ]:
```
import numpy as np
import sys

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
from zipfile import ZipFile
file_name = 'train.zip'

with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print('Done extracting train.zip')
```

```
Done extracting train.zip
```

In [ ]:
```
from zipfile import ZipFile
file_name = 'test.zip'

with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print('Done extracting test.zip')
```

```
Done extracting test.zip
```

In [ ]:
```
from zipfile import ZipFile
file_name = 'validation.zip'

with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print('Done extracting test.zip')
```

```
Done extracting test.zip
```

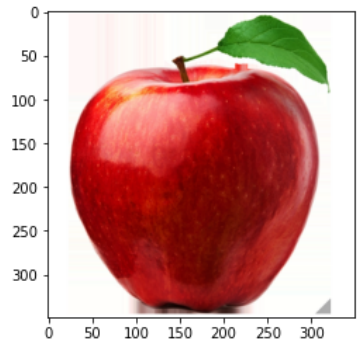
In [ ]:
```
from zipfile import ZipFile
file_name = 'test1.zip'

with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print('Done extracting test1.zip')
```

```
Done extracting test1.zip
```

In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import cv2
import matplotlib.pyplot as plot
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import RMSprop, Adam
import os
import PIL
```

In [ ]:
```
img =image.load_img('train/apple/apple_1.jpg')
```

In [ ]:
```
plot.imshow(img)
cv2.imread("train/apple/apple_1.JPG")

#the only part that uses cv2 actually so cv2 isn't actually required
#other than for the sole purpose of verification if the
```



In [ ]:
```
train =  ImageDataGenerator(rescale =  1/255)
validation= ImageDataGenerator(rescale =  1/255)
```

In [ ]:
```
train_dataset= train.flow_from_directory("train/",
                                         target_size = (200,200),
                                         batch_size=10,
                                         color_mode='rgb',
                                         class_mode ='categorical')

validation_dataset= validation.flow_from_directory("validation/",
                                         target_size = (200,200),
                                         batch_size=10,
                                         color_mode='rgb',
                                         class_mode ='categorical')
```

```
Found 240 images belonging to 4 classes.
Found 56 images belonging to 4 classes.
```

In [ ]:
```
validation_dataset= validation.flow_from_directory("test1/",
                                         target_size = (200,200),
                                         batch_size=10,
                                         color_mode='rgb',
                                         class_mode ='categorical')
```

```
Found 55 images belonging to 4 classes.
```

Making my model

In [ ]:
```
model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(64,(3,3), activation  ='relu', input_shape =(200,200,3)),
                                    tf.keras.layers.Conv2D(32,(3,3), activation  ='relu'),
                                    tf.keras.layers.MaxPool2D(2,2),
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(128,activation ="relu"),
                                    tf.keras.layers.Dropout(0.5),
                                    tf.keras.layers.Dense(4,activation ="softmax")])
```

In [ ]:
```
model.compile (loss = "categorical_crossentropy",
               optimizer = Adam(learning_rate=0.001),
                metrics =['accuracy'])

#here we decided to try the Adam optimization algorithm,
#best accuracy of 85.5% is achieved, train time of ~624s, with ~21s an epoch and ~2s a step
#we also retrained the model, using data from the test set as for our validation model
#train time of ~685s, with 23s an epoch and ~2 a step
#result of last run of training the model was
#loss: 0.0018 - accuracy: 1.0000 - val_loss: 6.7386e-05 - val_accuracy: 1.0000
#However, actual results translated only a result of 69.1% accuracy when putting the model to evaluate the test set
```

In [ ]:
```
model.compile (loss = "categorical_crossentropy",
               optimizer = RMSprop(learning_rate=0.001),
                metrics =['accuracy'])

#here we decided to try another optimization algorithm, which we found to yield a better accuracy overall
#best accuracy of 96.43% is achieved, train time of ~647s, ~23s an epoch and ~2s step
#actual results translated only a result of 92.7% accuracy when putting the model to evaluate the test set
#we also retrained the model, using data from the test set as for our validation model
#best accuracy of 92.73% is achieved, train time of ~720s, ~24s an epoch and ~2s step
#result of the last run of training the model is at epoch 3 where we stopped it as 100% accuracy is achieved
#loss: 0.0056 - accuracy: 1.0000 - val_loss: 1.6429e-04 - val_accuracy: 1.0000
#actual accuracy of test set achieved is 92.7% when putting the model to the test
```

In [ ]:
```
model_fit =model.fit(train_dataset,
                 steps_per_epoch= 10,
                 epochs =30,
                validation_data=validation_dataset)

#we found that just running 10 steps per epoch and 30 epochs is sufficient to reach a high enough degree of accuracy
#upon observing an 100% accuracy and validation accurate, we terminated the training to save time
```

```
Epoch 1/30
 2/10 [=====>.....................] - ETA: 16s - loss: 0.0028 - accuracy: 1.0000
/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Transparency expressed in bytes shoul
d be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
10/10 [==============================] - 24s 2s/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 8.8485e-05 - val_accuracy:
1.0000
Epoch 2/30
10/10 [==============================] - 24s 2s/step - loss: 0.0244 - accuracy: 0.9900 - val_loss: 0.0134 - val_accuracy: 1.000
0
Epoch 3/30
10/10 [==============================] - 24s 2s/step - loss: 0.0056 - accuracy: 1.0000 - val_loss: 1.6429e-04 - val_accuracy:
1.0000
Epoch 4/30
 4/10 [==========>...................] - ETA: 12s - loss: 1.9507 - accuracy: 0.8500
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-48-fcae1eef8eba> in <module>()
      2                  steps_per_epoch= 10,
      3                  epochs =30,
----> 4                 validation_data=validation_dataset)
      5
      6 #we found that just running 10 steps per epoch and 30 epochs is sufficient to reach a high enough degree of accuracy

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py in fit(self, x, y, batch_size, epochs, verbos
e, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validati
on_steps, validation_batch_size, validation_freq, max_queue_size, workers, use_multiprocessing)
   1181                 _r=1):
   1182               callbacks.on_train_batch_begin(step)
-> 1183               tmp_logs = self.train_function(iterator)
   1184               if data_handler.should_sync:
   1185                 context.async_wait()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in __call__(self, *args, **kwds)
    887
    888       with OptionalXlaContext(self._jit_compile):
--> 889         result = self._call(*args, **kwds)
    890
    891       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in _call(self, *args, **kwds)
    915       # In this case we have created variables on the first call, so we run the
    916       # defunned version which is guaranteed to never create variables.
--> 917       return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
    918     elif self._stateful_fn is not None:
    919       # Release the lock early so that multiple threads can perform the call

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in __call__(self, *args, **kwargs)
   3022        filtered_flat_args) = self._maybe_define_function(args, kwargs)
   3023     return graph_function._call_flat(
-> 3024         filtered_flat_args, captured_inputs=graph_function.captured_inputs)  # pylint: disable=protected-access
   3025
   3026   @property

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _call_flat(self, args, captured_inputs, cancellat
ion_manager)
   1959       # No tape is watching; skip to running the function.
   1960       return self._build_call_outputs(self._inference_function.call(
-> 1961           ctx, args, cancellation_manager=cancellation_manager))
   1962     forward_backward = self._select_forward_and_backward_functions(
   1963         args,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in call(self, ctx, args, cancellation_manager)
    594               inputs=args,
    595               attrs=attrs,
--> 596               ctx=ctx)
    597         else:
    598           outputs = execute.execute_with_cancellation(

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs,
ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

KeyboardInterrupt:
```

In [ ]:

```
validation_dataset.class_indices
```

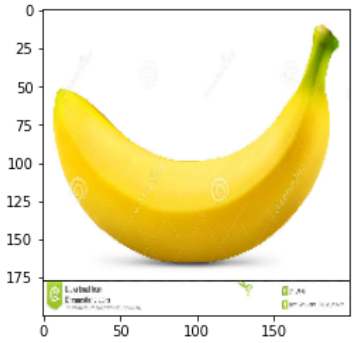
Out[ ]:

```
{'apple': 0, 'banana': 1, 'mixed': 2, 'orange': 3}
```

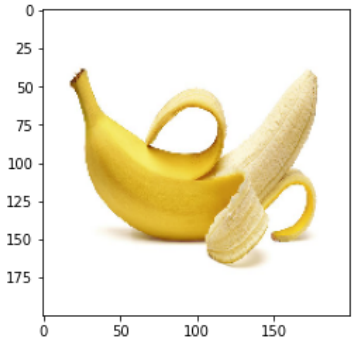
In [ ]:

```
dir_path= 'test/'

data = np.asarray(['apple','banana','mixed','orange'])
new_array= np.zeros((1,4))
#instantiates an empty numpy array
total = len(os.listdir(dir_path))
count = 0

for filename in os.listdir(dir_path):
    img = image.load_img(dir_path + filename, target_size=(200,200))
    img_array = image.img_to_array(img)
    img_batch = np.expand_dims(img_array, axis=0)
```

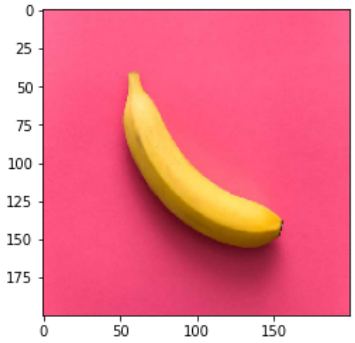
```
    prediction = model.predict(img_batch)
    pred = prediction[0]

    idx, = np.where(pred==max(pred))
    #
    # Recording test set as 2D array
    new_array = np.vstack([new_array,pred])

    # Display + keep track of no. of correct predictions
    if len(idx) >1:
        plot.imshow(img)
        plot.show()
        print(f"{filename:s} is unknown by model, error in prediction")
    else:
        j = idx[0]
        plot.imshow(img)
        plot.show()
        print(f"{filename:s} is {data[j]}")
        if data[j] in filename:
            count += 1

print(f"Accuracy of test set is {count/total:0.3f}")
# print(new_array)
```



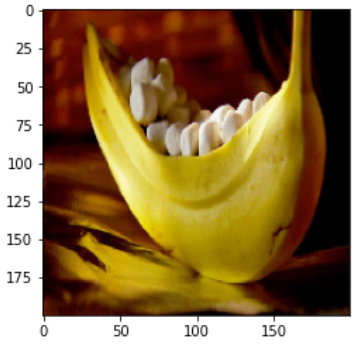
banana_79.jpg is banana



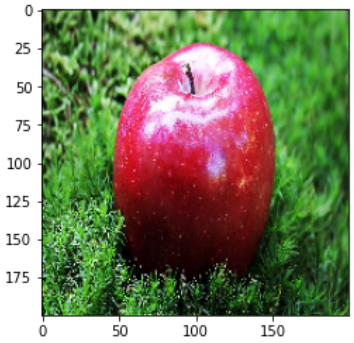
banana_82.jpg is banana



banana_88.jpg is apple

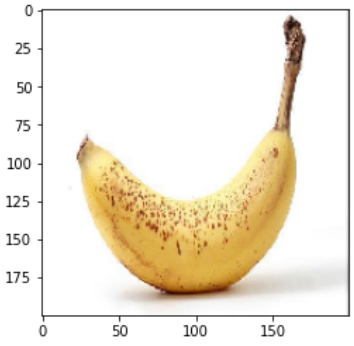


banana_83.jpg is orange

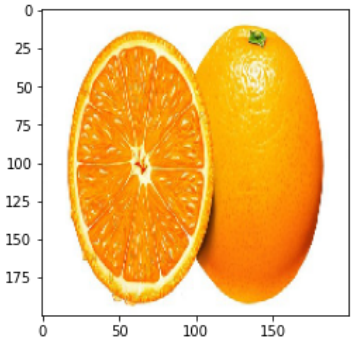


apple_91.jpg is apple

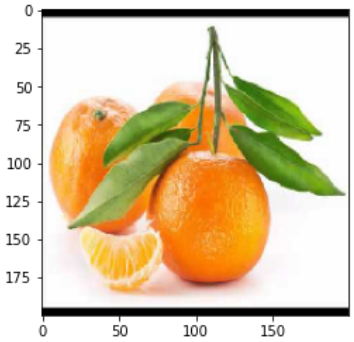
apple_77.jpg is orange



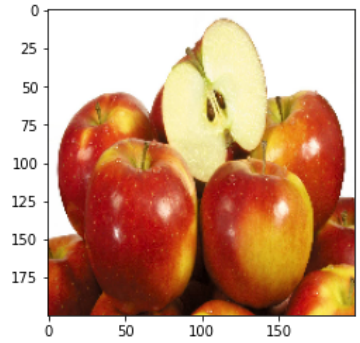
banana_84.jpg is banana



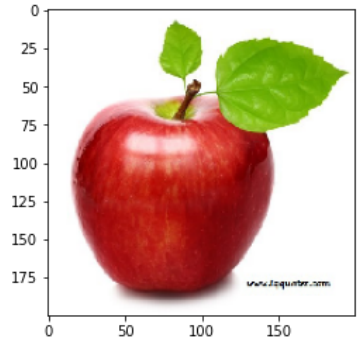
orange_78.jpg is orange



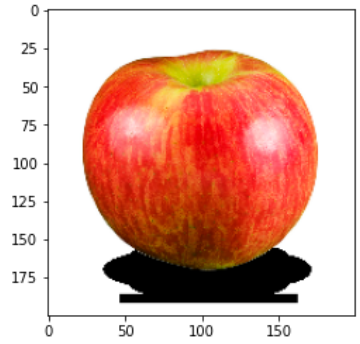
orange_86.jpg is orange



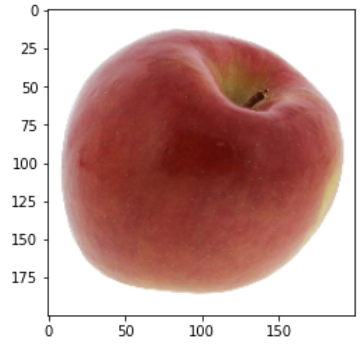
apple_95.jpg is apple



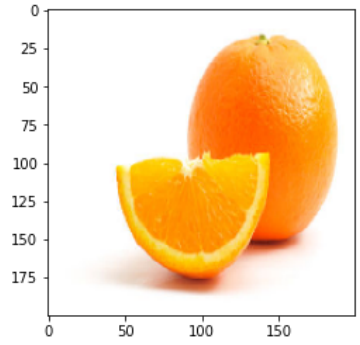
apple_87.jpg is apple



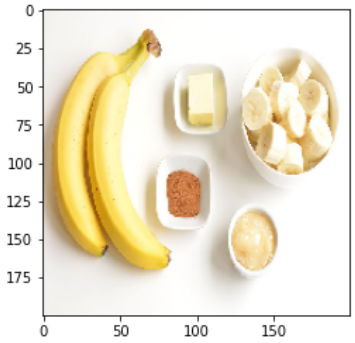
apple_92.jpg is apple



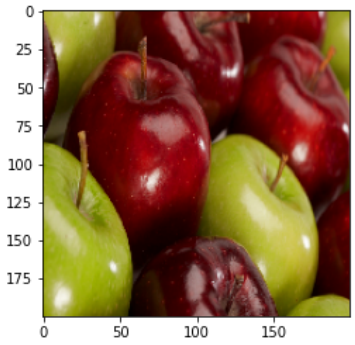
apple_90.jpg is apple



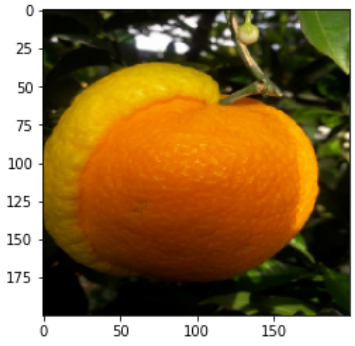
orange_92.jpg is orange



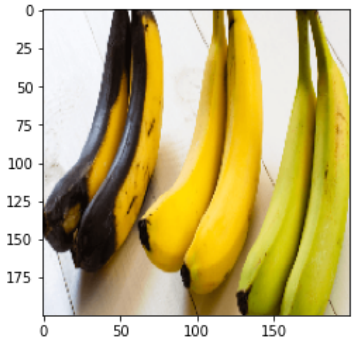
banana_91.jpg is banana



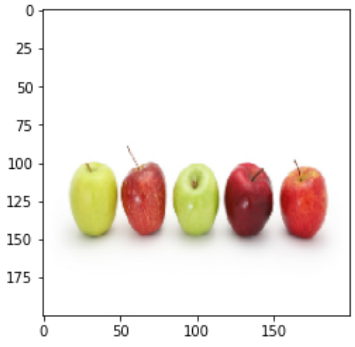
apple_84.jpg is apple



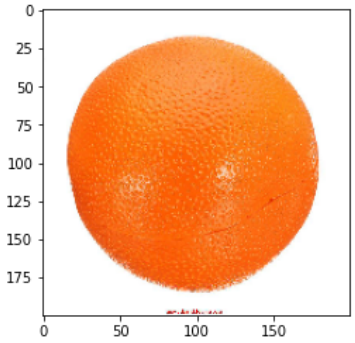
orange_85.jpg is orange



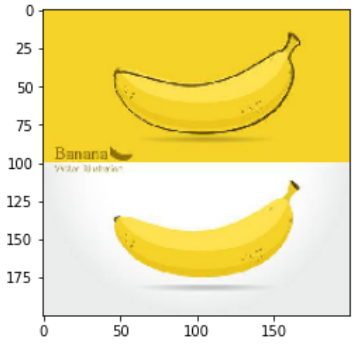
banana_87.jpg is banana



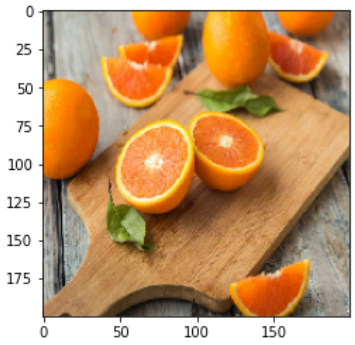
apple_86.jpg is apple



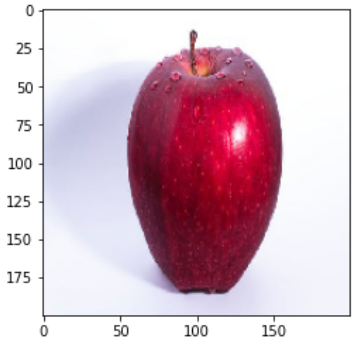
orange_83.jpg is orange



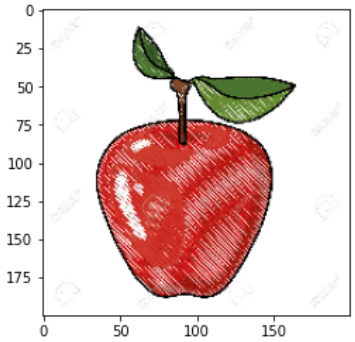
banana_93.jpg is banana



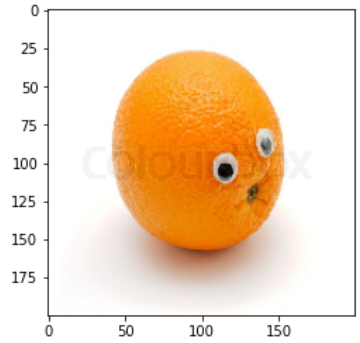
orange_77.jpg is orange



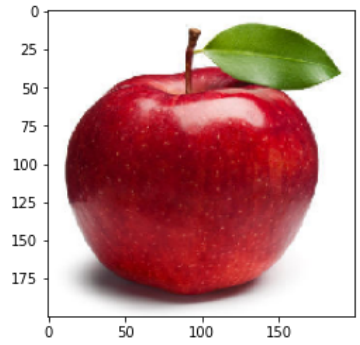
apple_93.jpg is apple



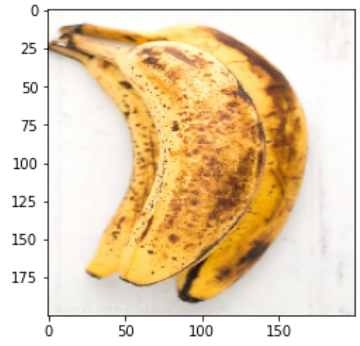
apple_88.jpg is apple



orange_91.jpg is orange

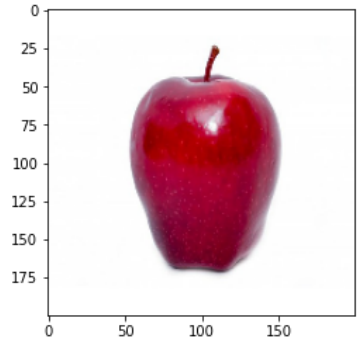


apple_89.jpg is apple

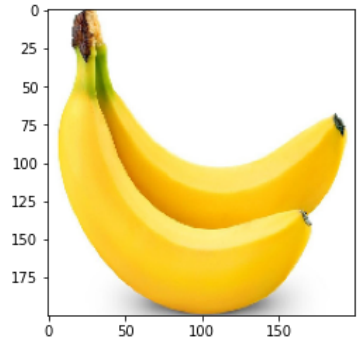


banana_86.jpg is banana

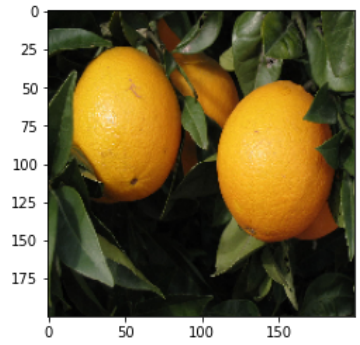
banana_90.jpg is banana



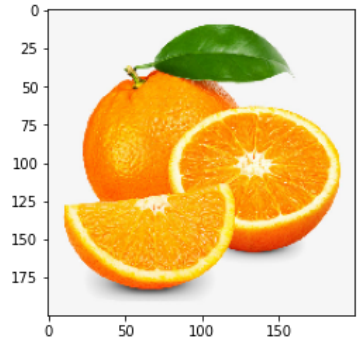
apple_80.jpg is apple



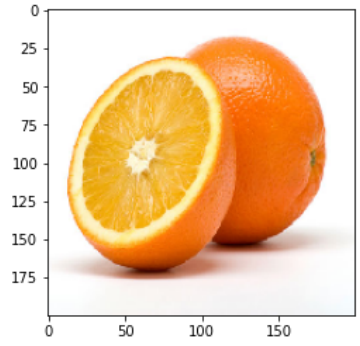
banana_81.jpg is banana



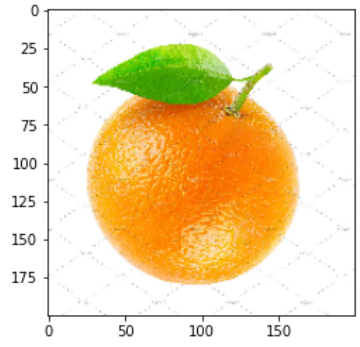
orange_89.jpg is orange



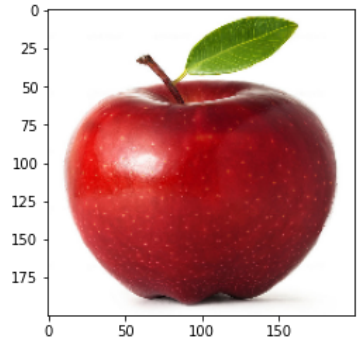
orange_82.jpg is orange



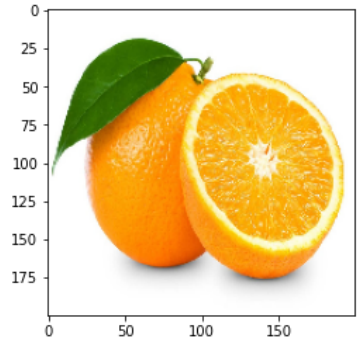
orange_84.jpg is orange



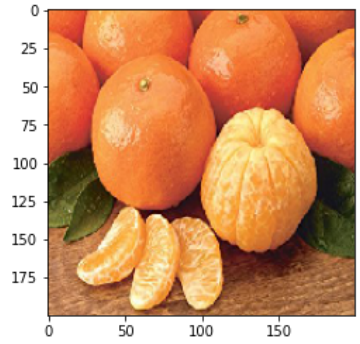
orange_95.jpg is orange



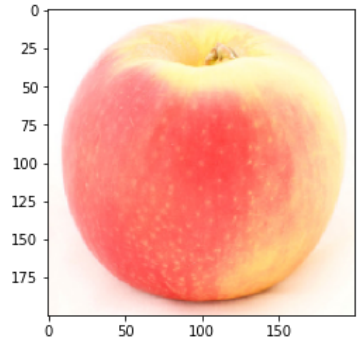
apple_81.jpg is apple



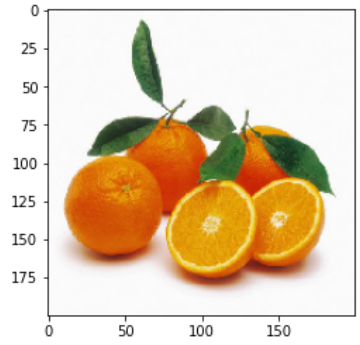
orange_80.jpg is orange



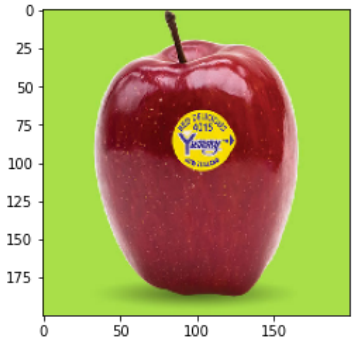
orange_90.jpg is orange



apple_78.jpg is apple

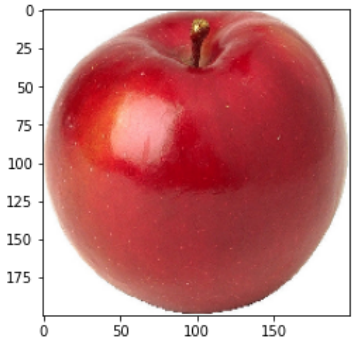


orange_87.jpg is orange

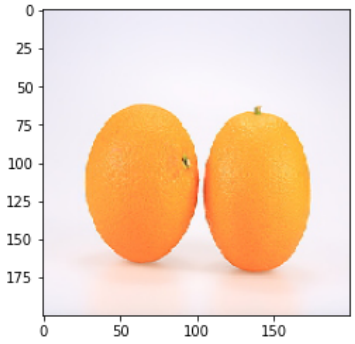


apple_85.jpg is apple

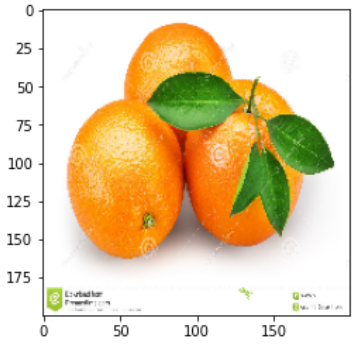
apple_94.jpg is apple



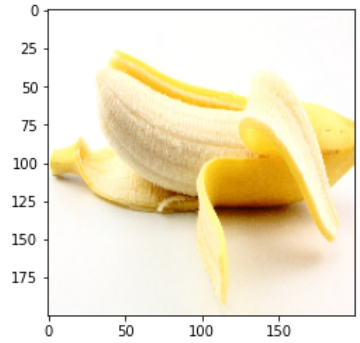
apple_83.jpg is apple



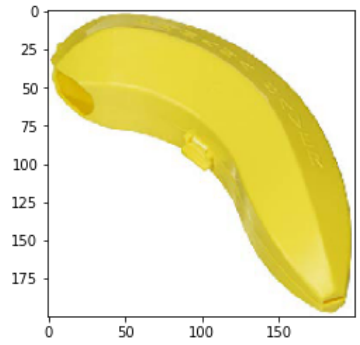
orange_93.jpg is orange



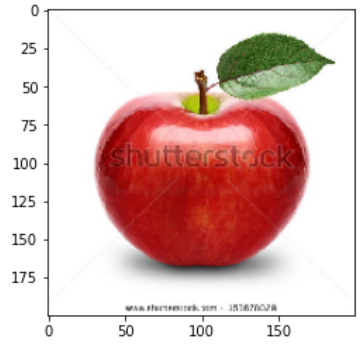
orange_81.jpg is orange



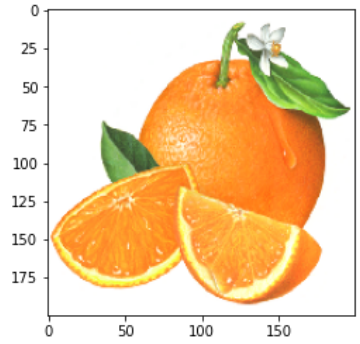
banana_85.jpg is banana



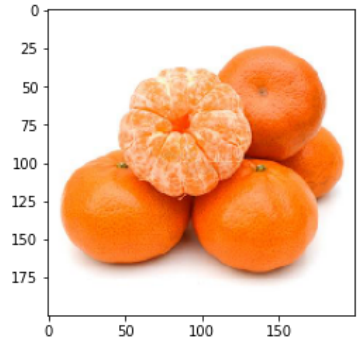
banana_94.jpg is banana



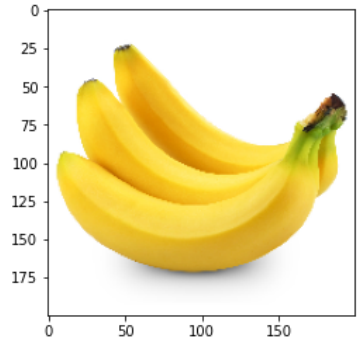
apple_82.jpg is apple



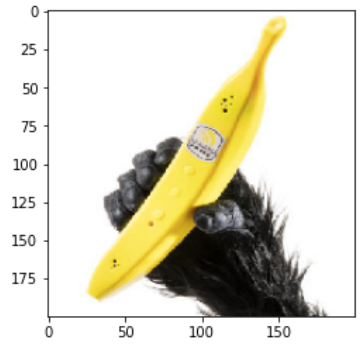
orange_94.jpg is orange



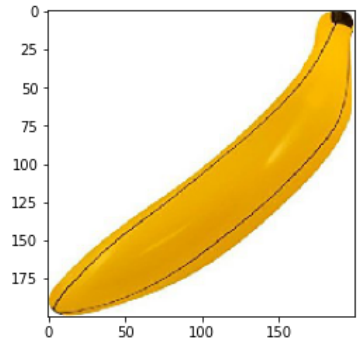
orange_79.jpg is orange



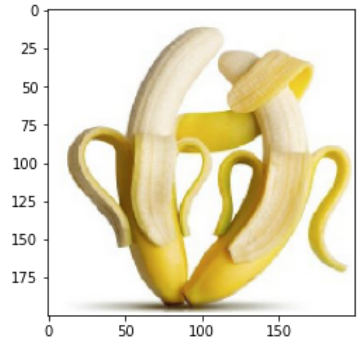
banana_89.jpg is orange



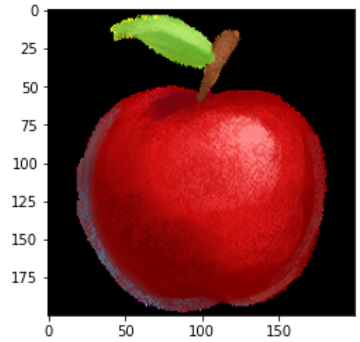
banana_77.jpg is banana



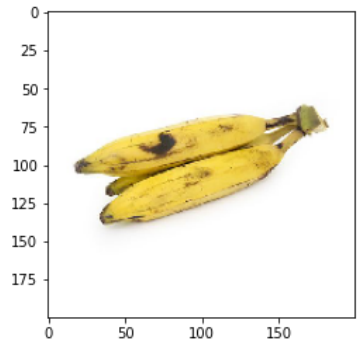
banana_92.jpg is banana



banana_78.jpg is banana



apple_79.jpg is apple



```
banana_80.jpg is banana
Accuracy of test set is 0.927
```

Model 2a: Adam and RMSprop optimization algorithms; 30 x 30 px

In [ ]:
```
from PIL import Image
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
from zipfile import ZipFile
file_name = 'train1.zip'

with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print('Done extracting train1.zip')
```

```
Done extracting train1.zip
```

In [ ]:
```
data=[]
data1=[]
for i in range(1,34):
    img=Image.open("train1/apple_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("apple")

for i in range(35,77):
    img=Image.open("train1/apple_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("apple")

for i in range(1,15):
    img=Image.open("train1/banana_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("banana")

for i in range(16,18):
    img=Image.open("train1/banana_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("banana")

for i in range(20,77):
    img=Image.open("train1/banana_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("banana")

for i in range(1,21):
    img=Image.open("train1/mixed_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("mixed")

for i in range(1,39):
    img=Image.open("train1/orange_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("orange")

for i in range(40,45):
    img=Image.open("train1/orange_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
```

```
    array = np.array(img_re)
    data.append(array)
    data1.append("orange")

for i in range(46,65):
    img=Image.open("train1/orange_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("orange")

for i in range(67,77):
    img=Image.open("train1/orange_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("orange")

feature=np.array(data)
label=np.array(data1)

print(feature)
print(label)
feature.shape
```

```
/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Transparency expressed in bytes shoul
d be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
[[[[255 255 255]
   [255 255 255]
   [255 249 249]
   ...
   [254 253 253]
   [255 255 255]
   [255 255 255]]

  [[255 255 255]
   [255 255 255]
   [255 248 248]
   ...
   [254 252 253]
   [255 255 255]
   [255 255 255]]

  [[255 255 255]
   [255 255 255]
   [255 248 248]
   ...
   [254 254 252]
   [255 255 255]
   [255 255 255]]

  ...

  [[255 255 255]
   [255 255 255]
   [255 254 251]
   ...
   [255 255 254]
   [255 255 255]
   [255 255 255]]

  [[255 255 255]
   [255 255 255]
   [255 254 251]
   ...
   [225 225 224]
   [255 255 255]
   [255 255 255]]

  [[255 255 255]
   [255 255 255]
   [255 254 252]
   ...
   [212 212 213]
   [255 255 255]
   [255 255 255]]]


 [[[255 255 255]
   [255 255 255]
   [255 255 255]
   ...
   [255 255 255]
   [255 255 255]
   [255 255 255]]

  [[255 255 255]
   [255 255 255]
   [255 255 255]
   ...
   [255 255 255]
   [255 255 255]
   [255 255 255]]

  [[255 255 255]
   [255 255 255]
```

```
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]]


[[[168  14  37]
  [214  31  83]
  [233  56 126]
  ...
  [248 122  97]
  [241 102  26]
  [169   7   0]]

 [[174   1   3]
  [193   0   0]
  [211   3  12]
  ...
  [240  23  34]
  [222  19  29]
  [158   1   1]]

 [[182   1   3]
  [208   1   8]
  [237  23  68]
  ...
  [252  71  62]
  [229  16  15]
  [165   0   0]]

 ...

 [[193   7   1]
  [166   0   0]
  [158   4   2]
  ...
  [254 118  18]
  [253 183  77]
  [252 208 128]]

 [[142   8  17]
  [136  42  70]
  [185  55  30]
  ...
  [253 154  27]
  [251 177  90]
  [251 193 143]]

 [[120  49 111]
  [159  89 152]
  [201  48   9]
  ...
  [251 110  21]
  [251 144  84]
  [251 169 154]]]


...


[[[248 247 252]
  [248 247 252]
  [248 247 252]
  ...
  [246 245 250]
  [246 245 250]
  [246 245 250]]

 [[248 247 252]
  [248 247 252]
  [248 247 252]
  ...
```

```
  [247 245 250]
  [246 245 250]
  [246 245 250]]

 [[248 247 253]
  [248 247 252]
  [248 248 253]
  ...
  [247 245 250]
  [247 245 250]
  [246 245 250]]

 ...

 [[245 243 246]
  [246 241 245]
  [245 240 244]
  ...
  [252 252 253]
  [252 252 254]
  [251 251 253]]

 [[245 243 246]
  [245 243 246]
  [245 241 245]
  ...
  [252 252 254]
  [251 251 253]
  [250 250 252]]

 [[245 243 246]
  [245 243 246]
  [244 242 245]
  ...
  [250 250 252]
  [249 249 251]
  [248 248 250]]]


[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...

 [[239 229 219]
  [237 226 214]
  [233 220 207]
  ...
  [226 203 171]
  [229 210 183]
  [233 217 193]]

 [[245 236 228]
  [242 233 224]
  [231 221 211]
  ...
  [229 209 180]
  [232 215 190]
  [234 219 196]]

 [[247 240 233]
  [247 239 231]
  [247 238 229]
  ...
  [234 218 194]
  [237 222 201]
  [238 225 206]]]


[[[255 152  46]
  [246 115  11]
  [232  91   0]
  ...
  [188  34   1]
  [195  34   1]
  [193  31   1]]

 [[255 147  18]
```

```
  [249 118   8]
  [236  88   1]
  ...
  [198  34   1]
  [204  34   1]
  [191  25   1]]

 [[254 151   9]
  [253 127   3]
  [236  75   1]
  ...
  [203  29   1]
  [183  19   1]
  [123   8   2]]

 ...

 [[189  54   2]
  [ 88  11   2]
  [197  52   1]
  ...
  [233 123   5]
  [217 100   4]
  [195  76   3]]

 [[202  56   1]
  [ 88  12   2]
  [177  42   1]
  ...
  [227 118   6]
  [207  90   3]
  [184  67   4]]

 [[208  56   2]
  [ 85  14   3]
  [147  30   2]
  ...
  [216 102   3]
  [197  81   3]
  [176  63   7]]]]
['apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple'
 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple'
 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple'
 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple'
 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple'
 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple'
 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple'
 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple' 'apple'
 'apple' 'apple' 'apple' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana' 'banana'
 'banana' 'banana' 'banana' 'banana' 'mixed' 'mixed' 'mixed' 'mixed'
 'mixed' 'mixed' 'mixed' 'mixed' 'mixed' 'mixed' 'mixed' 'mixed' 'mixed'
 'mixed' 'mixed' 'mixed' 'mixed' 'mixed' 'mixed' 'mixed' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange']
```

Out[ ]:
```
(240, 30, 30, 3)
```

In [ ]:
```
x_train=np.reshape(feature,(feature.shape[0],30,30,3))
x_train=x_train/255
```

In [ ]:
```
encoder=LabelEncoder()
encoder.fit(label)
labels=encoder.transform(label)
y_train=tf.keras.utils.to_categorical(labels)
```

In [ ]:
```
model=tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation="relu", input_shape=(30, 30, 3)))
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation="relu"))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation="relu"))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(4, activation="softmax"))
```

In [ ]:
```
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
#Epoch 30/30
# 22/22 [==============================] - 1s 42ms/step - loss: 0.0221 - accuracy: 0.9907 - val_loss: 0.6493 - val_accuracy: 0.
#Actual accuracy of test set is 94.5%
```

In [ ]:
```
model.compile(loss="categorical_crossentropy", optimizer="RMSprop", metrics=["accuracy"])
# Epoch 30/30
# 22/22 [==============================] - 1s 45ms/step - loss: 0.0515 - accuracy: 0.9815 - val_loss: 0.7634 - val_accuracy: 0.
#Actual accuracy of test set is 87.3%
```

In [ ]:
```
model.fit(x=x_train, y=y_train, batch_size=10, epochs=30, validation_split=0.1)
```

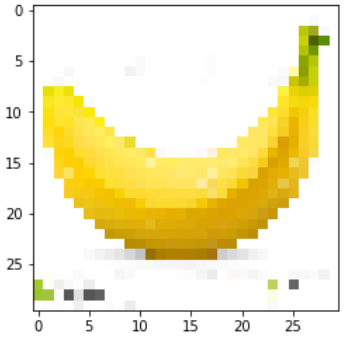
```
Epoch 1/30
22/22 [==============================] - 2s 51ms/step - loss: 1.1880 - accuracy: 0.5324 - val_loss: 1.1640 - val_accuracy: 0.12
50
Epoch 2/30
22/22 [==============================] - 1s 43ms/step - loss: 0.7352 - accuracy: 0.7176 - val_loss: 0.3245 - val_accuracy: 0.91
67
Epoch 3/30
22/22 [==============================] - 1s 44ms/step - loss: 0.4719 - accuracy: 0.8333 - val_loss: 0.7025 - val_accuracy: 0.83
33
Epoch 4/30
22/22 [==============================] - 1s 43ms/step - loss: 0.4209 - accuracy: 0.8426 - val_loss: 0.3343 - val_accuracy: 0.95
83
Epoch 5/30
22/22 [==============================] - 1s 44ms/step - loss: 0.3911 - accuracy: 0.8611 - val_loss: 1.0890 - val_accuracy: 0.58
33
Epoch 6/30
22/22 [==============================] - 1s 43ms/step - loss: 0.2740 - accuracy: 0.9074 - val_loss: 0.1154 - val_accuracy: 0.95
83
Epoch 7/30
22/22 [==============================] - 1s 45ms/step - loss: 0.2601 - accuracy: 0.9167 - val_loss: 0.6282 - val_accuracy: 0.79
17
Epoch 8/30
22/22 [==============================] - 1s 43ms/step - loss: 0.1813 - accuracy: 0.9537 - val_loss: 0.3131 - val_accuracy: 0.95
83
Epoch 9/30
22/22 [==============================] - 1s 45ms/step - loss: 0.2097 - accuracy: 0.9167 - val_loss: 1.8041 - val_accuracy: 0.54
17
Epoch 10/30
22/22 [==============================] - 1s 44ms/step - loss: 0.1567 - accuracy: 0.9352 - val_loss: 0.8466 - val_accuracy: 0.87
50
Epoch 11/30
22/22 [==============================] - 1s 44ms/step - loss: 0.1102 - accuracy: 0.9583 - val_loss: 0.6041 - val_accuracy: 0.83
33
Epoch 12/30
22/22 [==============================] - 1s 44ms/step - loss: 0.1485 - accuracy: 0.9444 - val_loss: 0.3974 - val_accuracy: 0.91
67
Epoch 13/30
22/22 [==============================] - 1s 43ms/step - loss: 0.1272 - accuracy: 0.9444 - val_loss: 1.4730 - val_accuracy: 0.66
67
Epoch 14/30
22/22 [==============================] - 1s 44ms/step - loss: 0.0783 - accuracy: 0.9676 - val_loss: 0.7081 - val_accuracy: 0.91
67
Epoch 15/30
22/22 [==============================] - 1s 44ms/step - loss: 0.0886 - accuracy: 0.9815 - val_loss: 0.5897 - val_accuracy: 0.95
83
Epoch 16/30
22/22 [==============================] - 1s 43ms/step - loss: 0.1227 - accuracy: 0.9722 - val_loss: 1.4943 - val_accuracy: 0.83
33
Epoch 17/30
22/22 [==============================] - 1s 42ms/step - loss: 0.0231 - accuracy: 0.9907 - val_loss: 1.3014 - val_accuracy: 0.79
17
Epoch 18/30
22/22 [==============================] - 1s 44ms/step - loss: 0.1772 - accuracy: 0.9491 - val_loss: 1.4631 - val_accuracy: 0.75
00
Epoch 19/30
22/22 [==============================] - 1s 43ms/step - loss: 0.0332 - accuracy: 0.9861 - val_loss: 0.4290 - val_accuracy: 0.95
83
Epoch 20/30
22/22 [==============================] - 1s 43ms/step - loss: 0.0555 - accuracy: 0.9815 - val_loss: 1.1392 - val_accuracy: 0.79
17
Epoch 21/30
22/22 [==============================] - 1s 43ms/step - loss: 0.0297 - accuracy: 0.9907 - val_loss: 1.4715 - val_accuracy: 0.83
33
Epoch 22/30
22/22 [==============================] - 1s 44ms/step - loss: 0.1062 - accuracy: 0.9630 - val_loss: 1.0123 - val_accuracy: 0.87
50
Epoch 23/30
22/22 [==============================] - 1s 44ms/step - loss: 0.0282 - accuracy: 0.9861 - val_loss: 0.9399 - val_accuracy: 0.87
50
Epoch 24/30
22/22 [==============================] - 1s 44ms/step - loss: 0.0956 - accuracy: 0.9630 - val_loss: 1.2002 - val_accuracy: 0.83
33
Epoch 25/30
22/22 [==============================] - 1s 44ms/step - loss: 0.0108 - accuracy: 1.0000 - val_loss: 0.9764 - val_accuracy: 0.87
50
Epoch 26/30
22/22 [==============================] - 1s 44ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.8680 - val_accuracy: 0.91
67
Epoch 27/30
22/22 [==============================] - 1s 43ms/step - loss: 0.0546 - accuracy: 0.9815 - val_loss: 4.0163 - val_accuracy: 0.54
17
Epoch 28/30
22/22 [==============================] - 1s 43ms/step - loss: 0.0385 - accuracy: 0.9861 - val_loss: 1.5148 - val_accuracy: 0.83
33
Epoch 29/30
22/22 [==============================] - 1s 44ms/step - loss: 0.0768 - accuracy: 0.9861 - val_loss: 1.8154 - val_accuracy: 0.87
50
Epoch 30/30
```

```
22/22 [==============================] - 1s 45ms/step - loss: 0.0515 - accuracy: 0.9815 - val_loss: 0.7634 - val_accuracy: 0.91
67
```

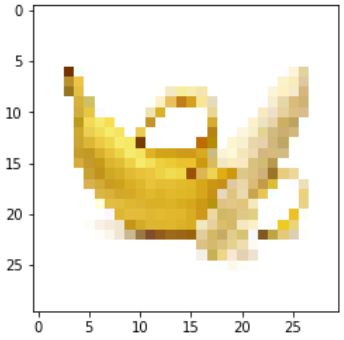
Out[ ]: <tensorflow.python.keras.callbacks.History at 0x7f7631c37cd0>

In [ ]:

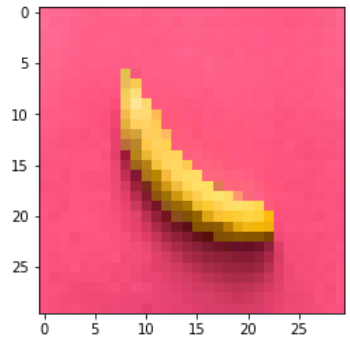
```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plot
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import RMSprop, Adam
import os
import PIL

dir_path= 'test/'

data = np.asarray(['apple','banana','mixed','orange'])
new_array= np.zeros((1,4))
#instantiates an empty numpy array
total = len(os.listdir(dir_path))
count = 0

for filename in os.listdir(dir_path):
    img = image.load_img(dir_path + filename, target_size=(30,30))
    img_array = image.img_to_array(img)
    img_batch = np.expand_dims(img_array, axis=0)
    prediction = model.predict(img_batch)
    pred = prediction[0]

    idx, = np.where(pred==max(pred))
    #
    # Recording test set as 2D array
    new_array = np.vstack([new_array,pred])

    # Display + keep track of no. of correct predictions
    if len(idx) >1:
        plot.imshow(img)
        plot.show()
        print(f"{filename:s} is unknown by model, error in prediction")
    else:
        j = idx[0]
        plot.imshow(img)
        plot.show()
        print(f"{filename:s} is {data[j]}")
        if data[j] in filename:
            count += 1

print(f"Accuracy of test set is {count/total:0.3f}")
# print(new_array)
```



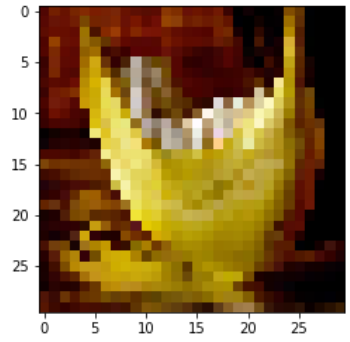
banana_79.jpg is banana



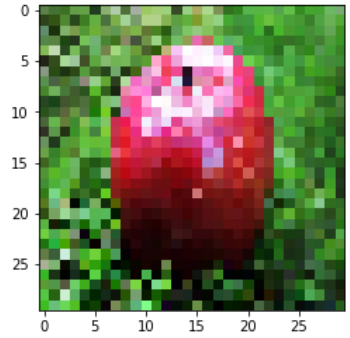
banana_82.jpg is banana



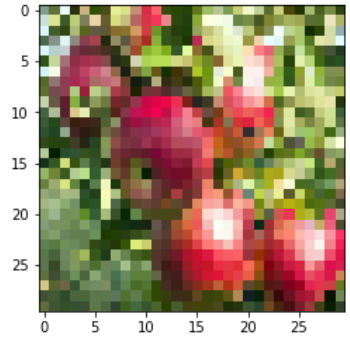
```
banana_88.jpg is apple
```



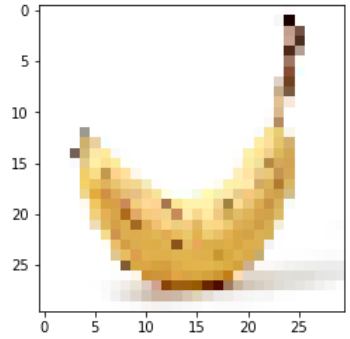
```
banana_83.jpg is banana
```



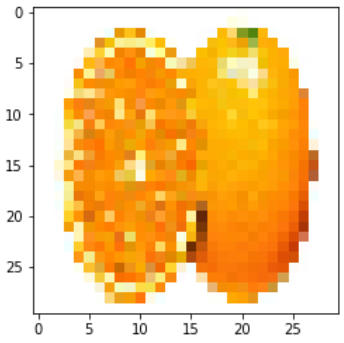
```
apple_91.jpg is apple
```



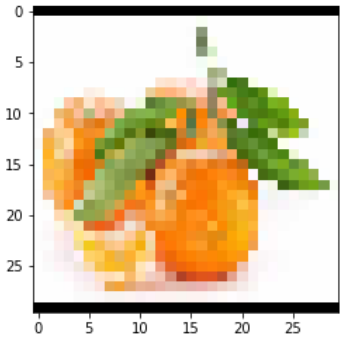
```
apple_77.jpg is apple
```



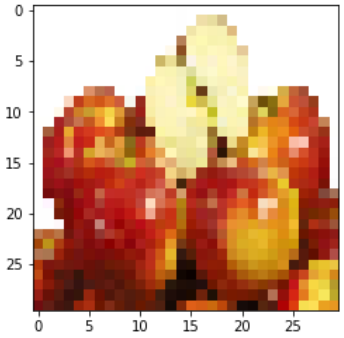
```
banana_84.jpg is banana
```



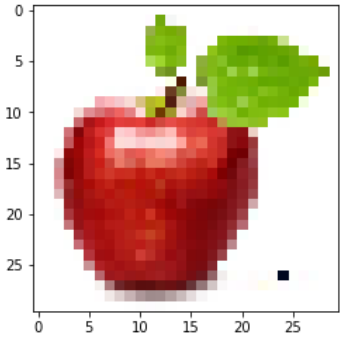
orange_78.jpg is orange



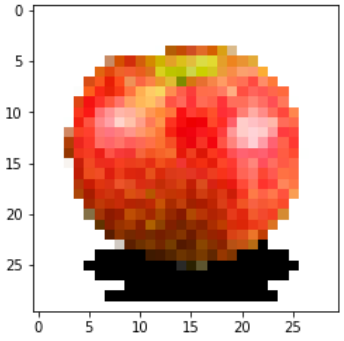
orange_86.jpg is orange



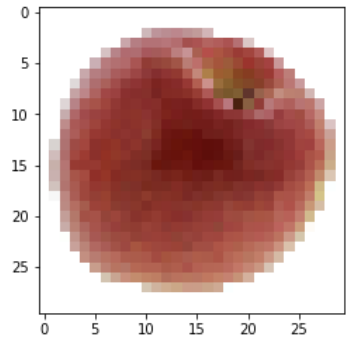
apple_95.jpg is orange



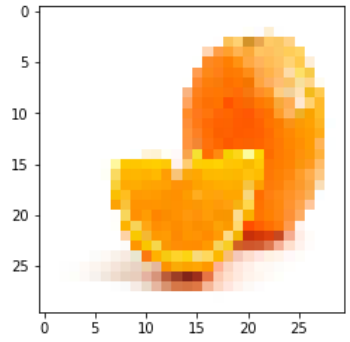
apple_87.jpg is apple



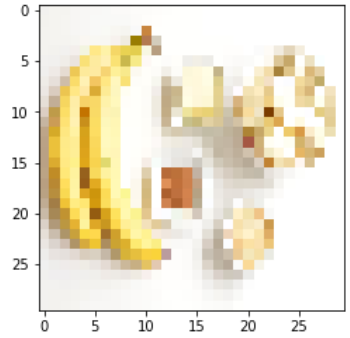
apple_92.jpg is apple



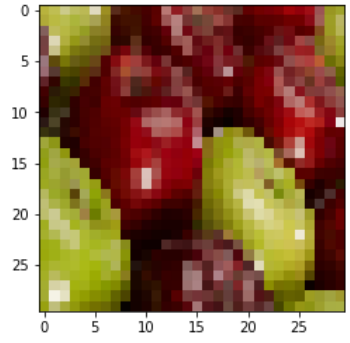
apple_90.jpg is apple



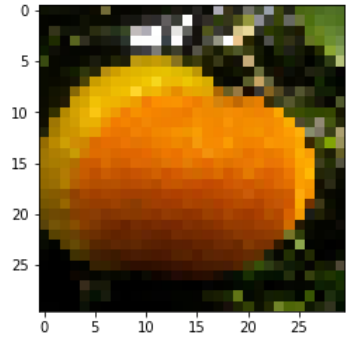
orange_92.jpg is orange



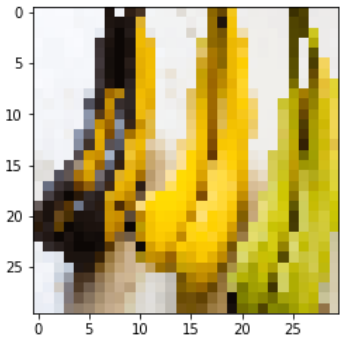
banana_91.jpg is banana



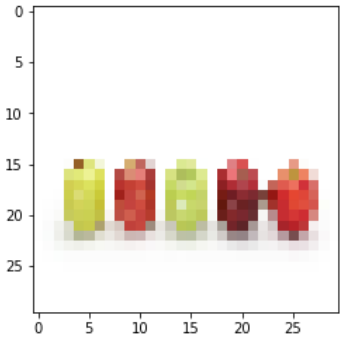
apple_84.jpg is apple



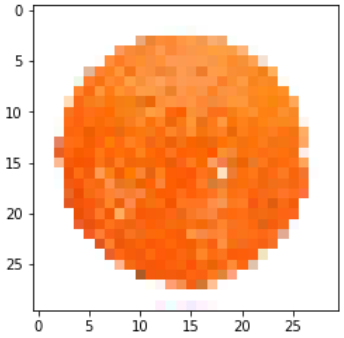
orange_85.jpg is orange



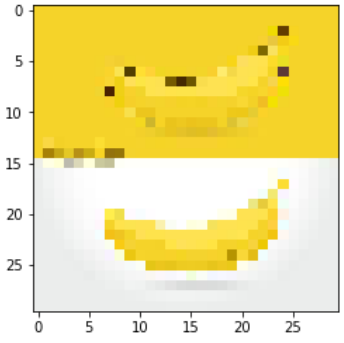
banana_87.jpg is banana



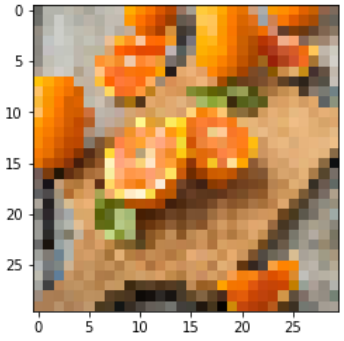
apple_86.jpg is banana



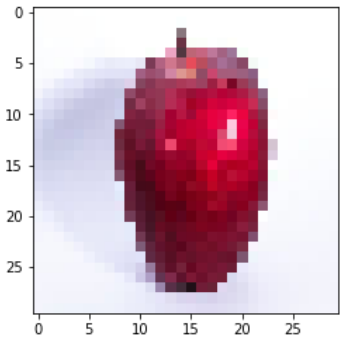
orange_83.jpg is orange



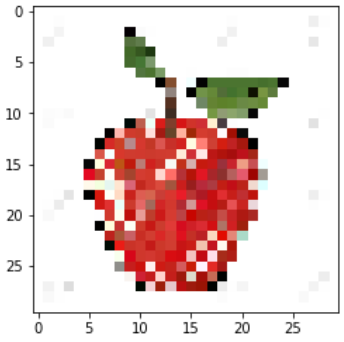
banana_93.jpg is banana



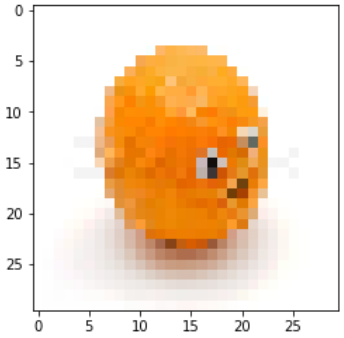
orange_77.jpg is banana



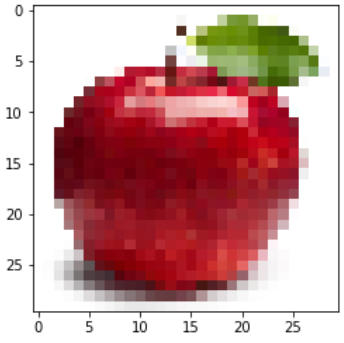
apple_93.jpg is apple



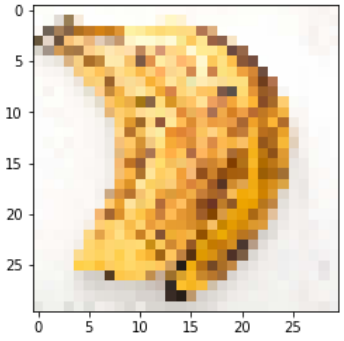
apple_88.jpg is apple



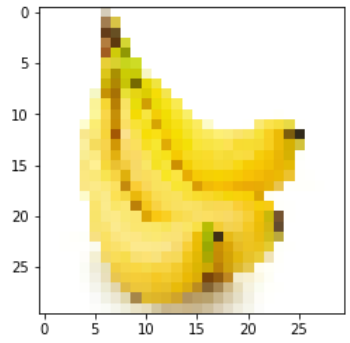
orange_91.jpg is orange



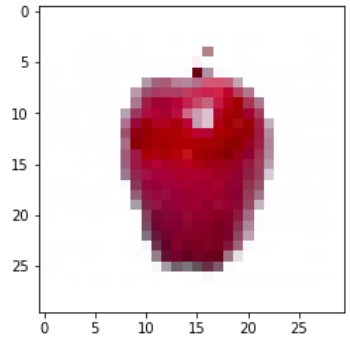
apple_89.jpg is apple



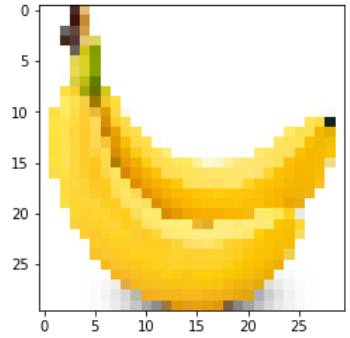
banana_86.jpg is orange



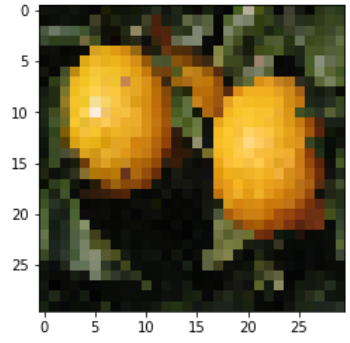
banana_90.jpg is banana



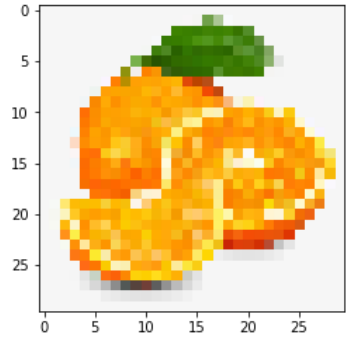
apple_80.jpg is apple



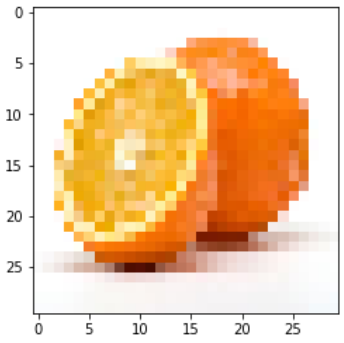
banana_81.jpg is banana



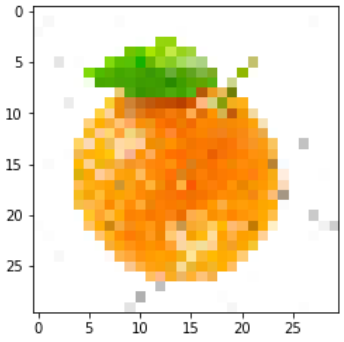
orange_89.jpg is orange



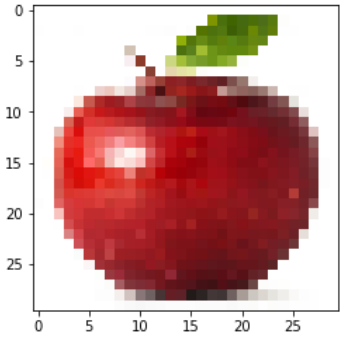
orange_82.jpg is orange



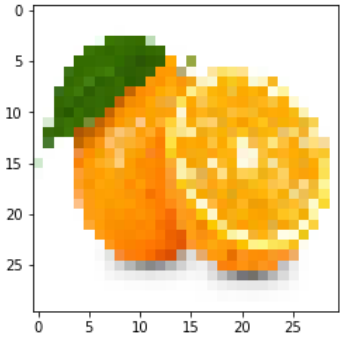
orange_84.jpg is orange



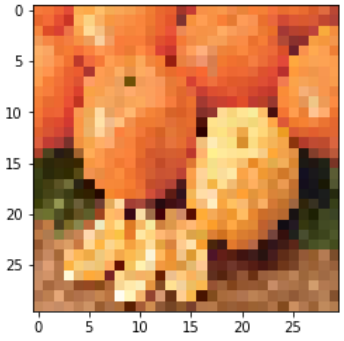
orange_95.jpg is orange



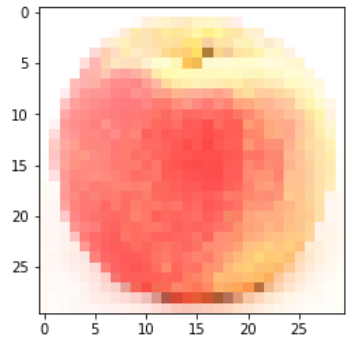
apple_81.jpg is apple



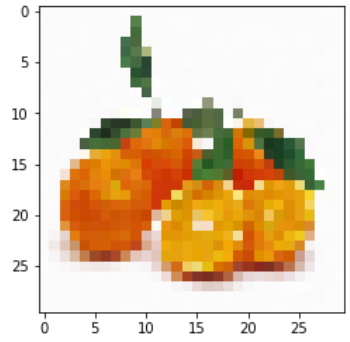
orange_80.jpg is orange



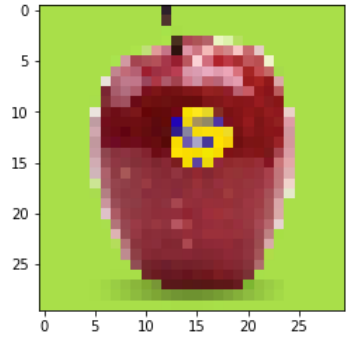
orange_90.jpg is banana



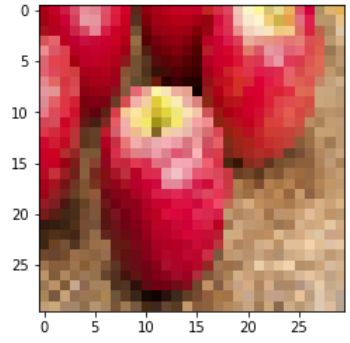
apple_78.jpg is apple



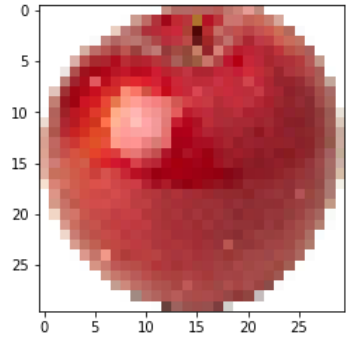
orange_87.jpg is orange



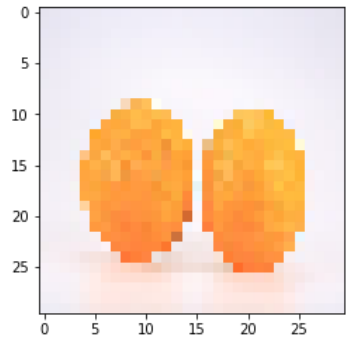
apple_85.jpg is apple



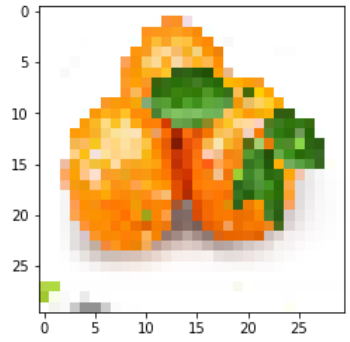
apple_94.jpg is apple



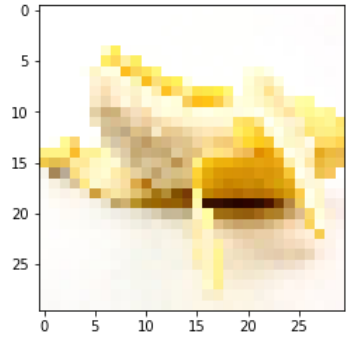
apple_83.jpg is apple



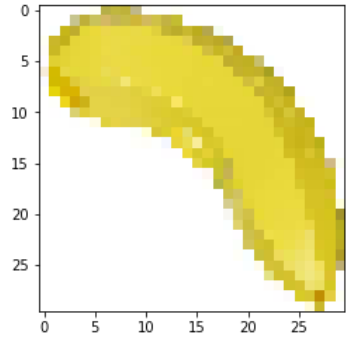
orange_93.jpg is orange



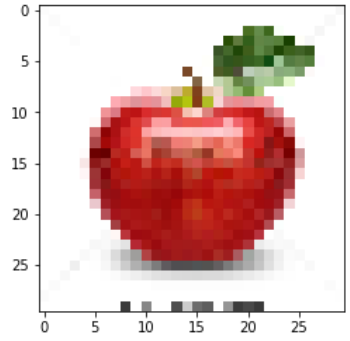
orange_81.jpg is orange



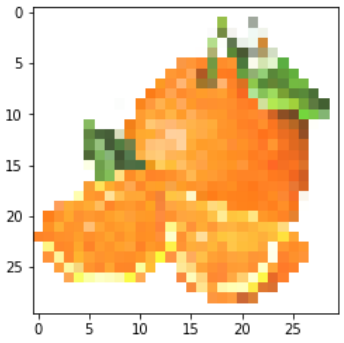
banana_85.jpg is banana



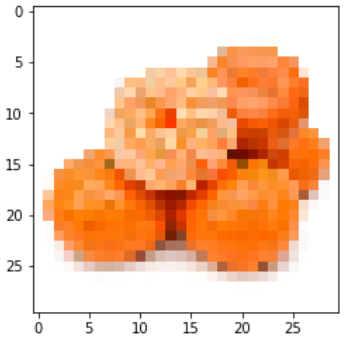
banana_94.jpg is banana



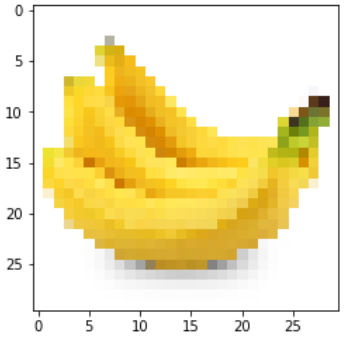
apple_82.jpg is apple



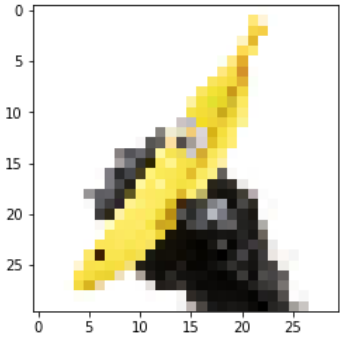
orange_94.jpg is orange



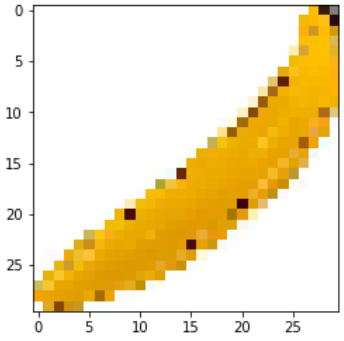
orange_79.jpg is orange



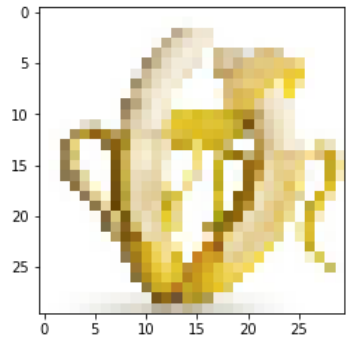
banana_89.jpg is orange



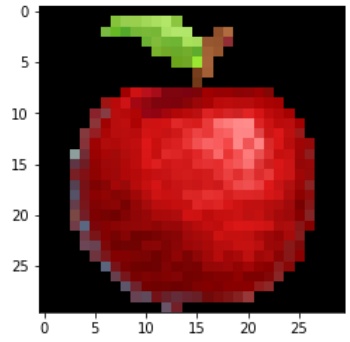
banana_77.jpg is banana



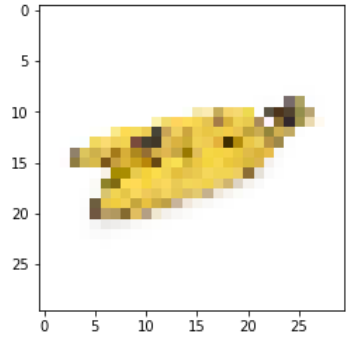
banana_92.jpg is banana



```
banana_78.jpg is banana
```



```
apple_79.jpg is apple
```



```
banana_80.jpg is banana
Accuracy of test set is 0.873
```

In [ ]:

```
from zipfile import ZipFile
file_name = 'test.zip'

with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print('Done extracting test.zip')
```

```
Done extracting test.zip
```

In [ ]:

```
data=[]
data1=[]
for i in range(77,96):
    img=Image.open("test/apple_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("apple")
for i in range(77,95):
    img=Image.open("test/banana_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("banana")

for i in range(77,88):
    img=Image.open("test/orange_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("orange")
for i in range(89,96):
    img=Image.open("test/orange_{0}.jpg".format (i)).convert("RGB")
    img_re=img.resize((30,30))
    array = np.array(img_re)
    data.append(array)
    data1.append("orange")
feature=np.array(data)
label=np.array(data1)
```

In [ ]:
```
x_test=np.reshape(feature,(feature.shape[0],30,30,3))
x_test=x_test/255
```

In [ ]:
```
encoder=LabelEncoder()
encoder.fit(label)
labels=encoder.transform(label)
y_test=tf.keras.utils.to_categorical(labels)
```

In [ ]:
```
loss, accuracy=model.evaluate(x=x_test, y=y_test)
```

```
2/2 [==============================] - 0s 27ms/step - loss: 0.7420 - accuracy: 0.8667
```

In [ ]:
```
print(loss,accuracy)
```

```
0.7420070171356201 0.8666666746139526
```

Model 2b: Adam and RMSprop optimization algorithms; 200 x 200 px

In [ ]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,Dropout,Flatten,Dense
```

In [ ]:
```
categories =['apple','banana','mixed','orange']

def process_img(file,numlist,features,label):
    for category in categories:

        for i in numlist:
            try:
                img= Image.open('%s/%s_%d.jpg' % (file,category,i)).convert('RGB')
                img=img.resize((200,200))
                img_array=np.array(img)
                features.append(img_array)
                label.append(category)

            except Exception as e:
                pass

        continue
```

In [ ]:
```
#set up train & test data
x_train=[]
y_train=[]
x_test=[]
y_test=[]

process_img('train1',range(1,77),x_train,y_train)
process_img('test',range(21,96),x_test,y_test)
```

```
/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Transparency expressed in bytes shoul
d be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
```

In [ ]:
```
#convert to numpy array
x_train=np.array(x_train)
y_train=np.array(y_train)
x_test=np.array(x_test)
y_test=np.array(y_test)
```

In [ ]:
```
x_train=np.reshape(x_train,(x_train.shape[0],200,200,3))
x_test=np.reshape(x_test,(x_test.shape[0],200,200,3))


x_train=x_train/255
x_test=x_test/255
```

In [ ]:
```
#one-hot encoding
encoder=LabelEncoder()
train=encoder.fit_transform(y_train)
test=encoder.fit_transform(y_test)

y_train=tf.keras.utils.to_categorical(train,4)
y_test=tf.keras.utils.to_categorical(test,4)
```

In [ ]:
```
model=Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=(200,200,3)))
model.add(Conv2D(32,(3,3),activation='relu'))
```

```
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4,activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='RMSprop',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 198, 198, 64)      1792
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 196, 196, 32)      18464
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 98, 98, 32)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 98, 98, 32)        0
_________________________________________________________________
flatten_9 (Flatten)          (None, 307328)            0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               39338112
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 4)                 516
=================================================================
Total params: 39,358,884
Trainable params: 39,358,884
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model=Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=(200,200,3)))
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4,activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
model.summary()

#Actual accuracy of test set is 0.945
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 198, 198, 64)      1792
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 196, 196, 32)      18464
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 98, 98, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 98, 98, 32)        0
_________________________________________________________________
flatten_7 (Flatten)          (None, 307328)            0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               39338112
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_15 (Dense)             (None, 4)                 516
=================================================================
Total params: 39,358,884
Trainable params: 39,358,884
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.fit(x_train,y_train,epochs=30,validation_split=0.1)
```

```
Epoch 1/30
7/7 [==============================] - 41s 6s/step - loss: 22.1578 - accuracy: 0.2870 - val_loss: 1.2297 - val_accuracy: 0.6667
Epoch 2/30
7/7 [==============================] - 39s 6s/step - loss: 1.3141 - accuracy: 0.3935 - val_loss: 1.0759 - val_accuracy: 0.5000
Epoch 3/30
7/7 [==============================] - 39s 6s/step - loss: 1.1933 - accuracy: 0.5741 - val_loss: 2.9888 - val_accuracy: 0.0000e
+00
Epoch 4/30
7/7 [==============================] - 39s 6s/step - loss: 1.3289 - accuracy: 0.5741 - val_loss: 8.2266 - val_accuracy: 0.0000e
+00
Epoch 5/30
7/7 [==============================] - 39s 6s/step - loss: 1.2484 - accuracy: 0.6435 - val_loss: 0.7581 - val_accuracy: 0.8750
Epoch 6/30
7/7 [==============================] - 39s 6s/step - loss: 0.8748 - accuracy: 0.7176 - val_loss: 2.4010 - val_accuracy: 0.5000
Epoch 7/30
7/7 [==============================] - 39s 6s/step - loss: 1.7339 - accuracy: 0.6343 - val_loss: 1.2332 - val_accuracy: 0.6250
Epoch 8/30
7/7 [==============================] - 39s 6s/step - loss: 0.4744 - accuracy: 0.8380 - val_loss: 1.3553 - val_accuracy: 0.7083
Epoch 9/30
7/7 [==============================] - 39s 6s/step - loss: 0.3017 - accuracy: 0.8704 - val_loss: 0.4281 - val_accuracy: 0.9167
Epoch 10/30
```

```
7/7 [==============================] - 39s 6s/step - loss: 0.2850 - accuracy: 0.9120 - val_loss: 2.7000 - val_accuracy: 0.4167
Epoch 11/30
7/7 [==============================] - 39s 6s/step - loss: 0.3160 - accuracy: 0.8935 - val_loss: 1.4529 - val_accuracy: 0.8333
Epoch 12/30
7/7 [==============================] - 39s 6s/step - loss: 0.1503 - accuracy: 0.9444 - val_loss: 3.3697 - val_accuracy: 0.5833
Epoch 13/30
7/7 [==============================] - 39s 6s/step - loss: 1.1601 - accuracy: 0.7824 - val_loss: 1.5009 - val_accuracy: 0.7917
Epoch 14/30
7/7 [==============================] - 39s 6s/step - loss: 0.2286 - accuracy: 0.9398 - val_loss: 4.2406 - val_accuracy: 0.3750
Epoch 15/30
7/7 [==============================] - 39s 6s/step - loss: 0.4991 - accuracy: 0.8657 - val_loss: 2.7280 - val_accuracy: 0.7500
Epoch 16/30
7/7 [==============================] - 39s 6s/step - loss: 0.9907 - accuracy: 0.8380 - val_loss: 1.6086 - val_accuracy: 0.5417
Epoch 17/30
7/7 [==============================] - 39s 6s/step - loss: 0.2549 - accuracy: 0.9213 - val_loss: 2.2039 - val_accuracy: 0.7500
Epoch 18/30
7/7 [==============================] - 39s 6s/step - loss: 0.1624 - accuracy: 0.9352 - val_loss: 1.5132 - val_accuracy: 0.8750
Epoch 19/30
7/7 [==============================] - 39s 6s/step - loss: 0.0603 - accuracy: 0.9815 - val_loss: 2.3174 - val_accuracy: 0.7917
Epoch 20/30
7/7 [==============================] - 39s 6s/step - loss: 0.0310 - accuracy: 0.9954 - val_loss: 3.2380 - val_accuracy: 0.7083
Epoch 21/30
7/7 [==============================] - 39s 6s/step - loss: 1.1582 - accuracy: 0.8981 - val_loss: 0.7867 - val_accuracy: 0.9167
Epoch 22/30
7/7 [==============================] - 39s 6s/step - loss: 2.3235 - accuracy: 0.7870 - val_loss: 1.5924 - val_accuracy: 0.7917
Epoch 23/30
7/7 [==============================] - 39s 6s/step - loss: 0.1956 - accuracy: 0.9167 - val_loss: 1.6880 - val_accuracy: 0.8333
Epoch 24/30
7/7 [==============================] - 39s 6s/step - loss: 0.0583 - accuracy: 0.9815 - val_loss: 1.5276 - val_accuracy: 0.8333
Epoch 25/30
7/7 [==============================] - 39s 6s/step - loss: 0.0359 - accuracy: 0.9861 - val_loss: 1.9324 - val_accuracy: 0.7917
Epoch 26/30
7/7 [==============================] - 39s 6s/step - loss: 0.0293 - accuracy: 0.9954 - val_loss: 1.8174 - val_accuracy: 0.8333
Epoch 27/30
7/7 [==============================] - 39s 6s/step - loss: 0.0575 - accuracy: 0.9815 - val_loss: 1.9868 - val_accuracy: 0.8333
Epoch 28/30
7/7 [==============================] - 39s 6s/step - loss: 0.5784 - accuracy: 0.8843 - val_loss: 0.3095 - val_accuracy: 0.9583
Epoch 29/30
7/7 [==============================] - 39s 6s/step - loss: 0.0678 - accuracy: 0.9907 - val_loss: 1.0882 - val_accuracy: 0.8750
Epoch 30/30
7/7 [==============================] - 39s 6s/step - loss: 0.0585 - accuracy: 0.9907 - val_loss: 1.5175 - val_accuracy: 0.8333
```

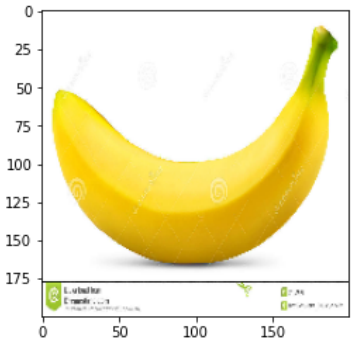
Out[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7f7632519f10>
```

In [ ]:
```
score=model.evaluate(x_test,y_test)
print('score=',score)
```

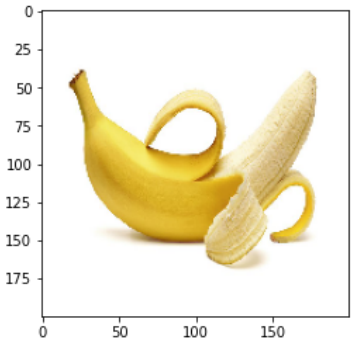
```
2/2 [==============================] - 2s 1s/step - loss: 1.0362 - accuracy: 0.8000
score= [1.0362112522125244, 0.800000011920929]
```

In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plot
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import RMSprop, Adam
import os
import PIL

dir_path= 'test/'

data = np.asarray(['apple','banana','mixed','orange'])
new_array= np.zeros((1,4))
#instantiates an empty numpy array
total = len(os.listdir(dir_path))
count = 0

for filename in os.listdir(dir_path):
    img = image.load_img(dir_path + filename, target_size=(200,200))
    img_array = image.img_to_array(img)
    img_batch = np.expand_dims(img_array, axis=0)
    prediction = model.predict(img_batch)
    pred = prediction[0]

    idx, = np.where(pred==max(pred))
    #
    # Recording test set as 2D array
    new_array = np.vstack([new_array,pred])

    # Display + keep track of no. of correct predictions
    if len(idx) >1:
        plot.imshow(img)
        plot.show()
        print(f"{filename:s} is unknown by model, error in prediction")
    else:
        j = idx[0]
        plot.imshow(img)
        plot.show()
        print(f"{filename:s} is {data[j]}")
        if data[j] in filename:
            count += 1
```

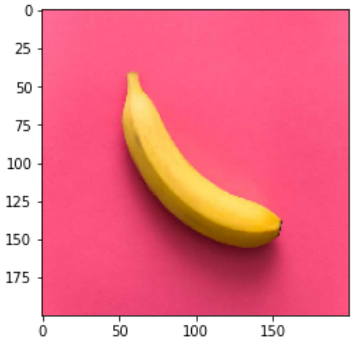
```
print(f"Accuracy of test set is {count/total:0.3f}")
# print(new_array)
```



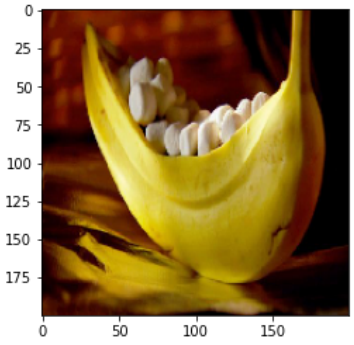
banana_79.jpg is banana



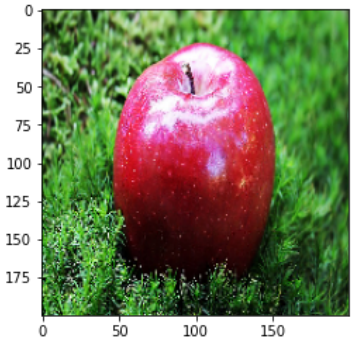
banana_82.jpg is banana



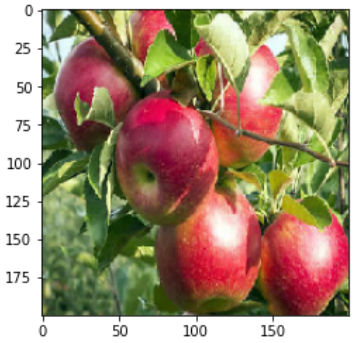
banana_88.jpg is apple



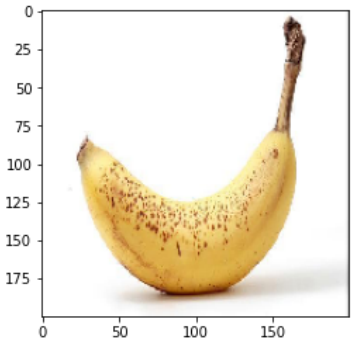
banana_83.jpg is banana



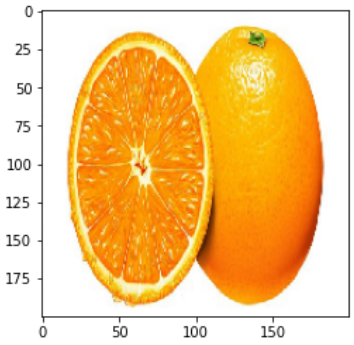
apple_91.jpg is apple



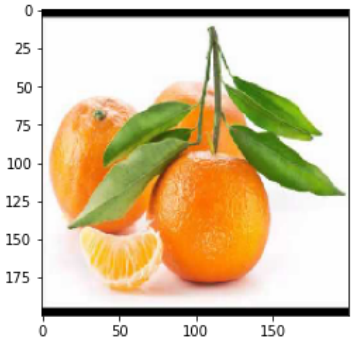
apple_77.jpg is apple



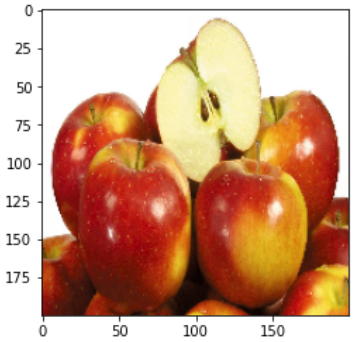
banana_84.jpg is banana



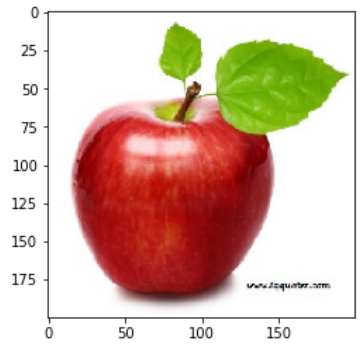
orange_78.jpg is orange



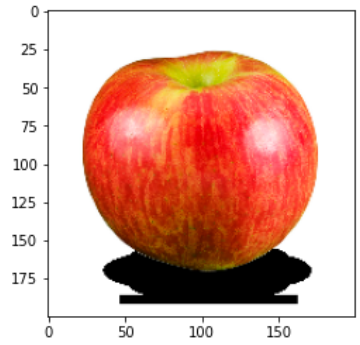
orange_86.jpg is banana



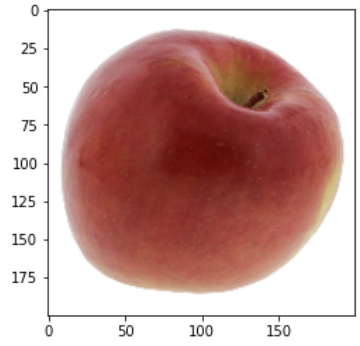
apple_95.jpg is apple



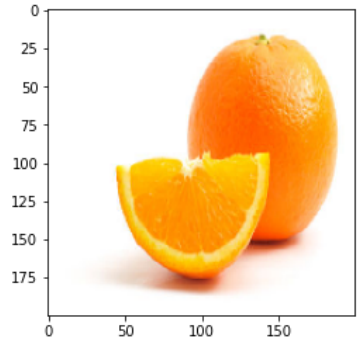
```
apple_87.jpg is apple
```



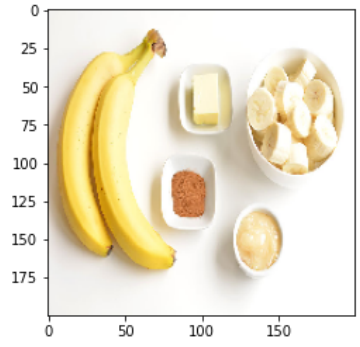
```
apple_92.jpg is apple
```



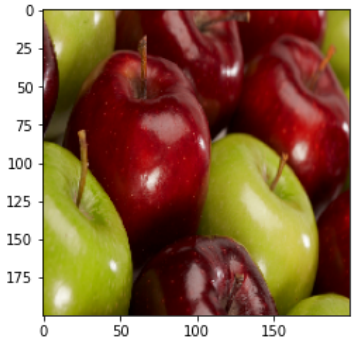
```
apple_90.jpg is apple
```



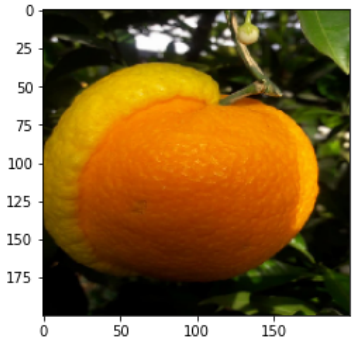
```
orange_92.jpg is orange
```



```
banana_91.jpg is banana
```

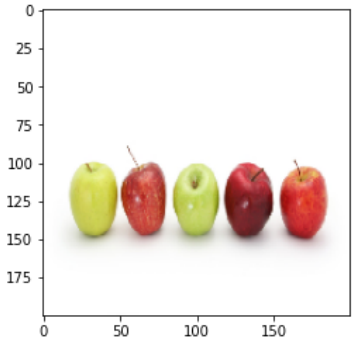


apple_84.jpg is apple

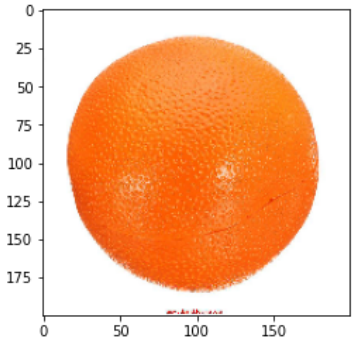


orange_85.jpg is orange

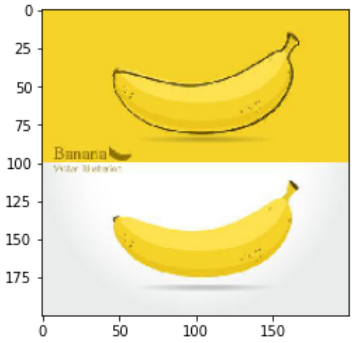
banana_87.jpg is banana



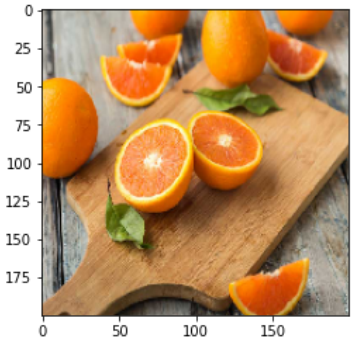
apple_86.jpg is apple



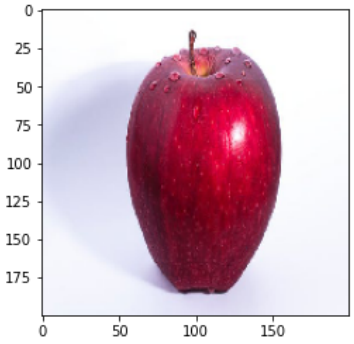
orange_83.jpg is orange



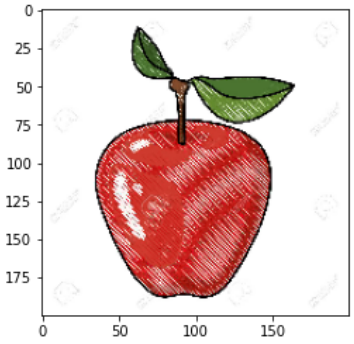
banana_93.jpg is banana



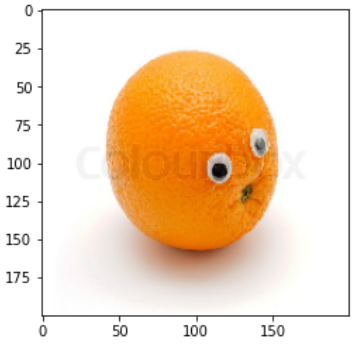
orange_77.jpg is banana



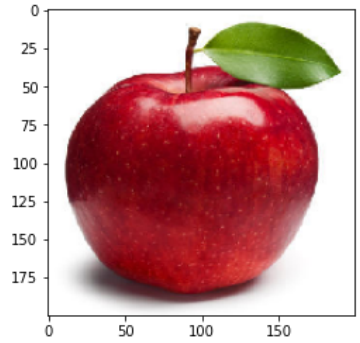
apple_93.jpg is apple



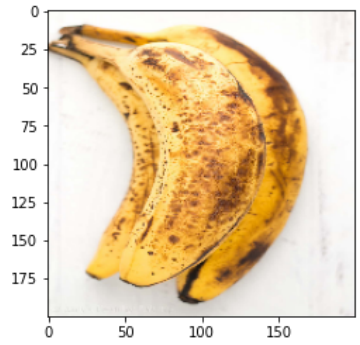
apple_88.jpg is apple



orange_91.jpg is orange

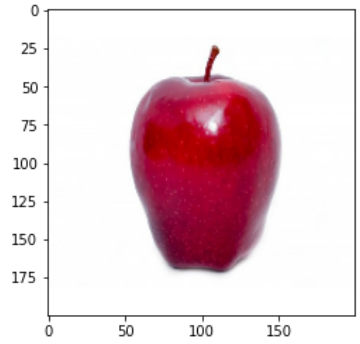


apple_89.jpg is apple

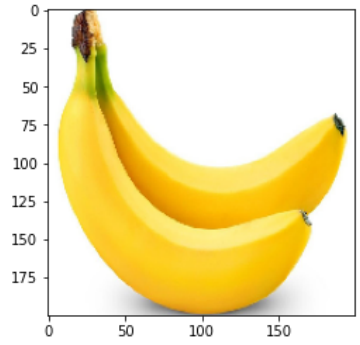


banana_86.jpg is banana

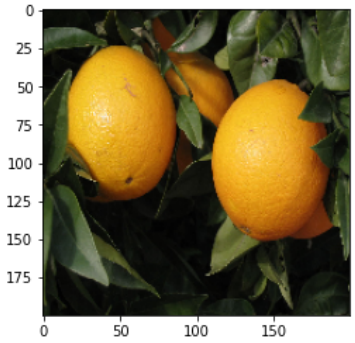
banana_90.jpg is banana



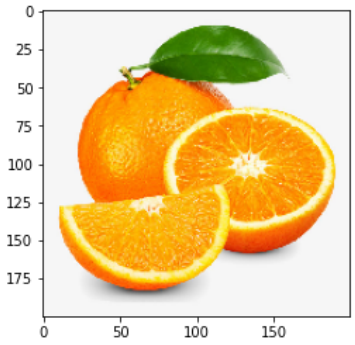
apple_80.jpg is apple



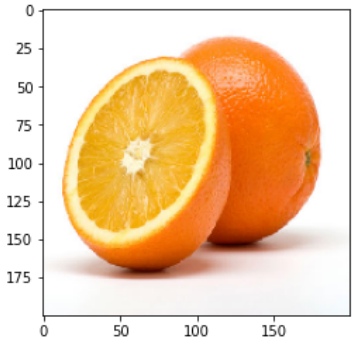
banana_81.jpg is banana



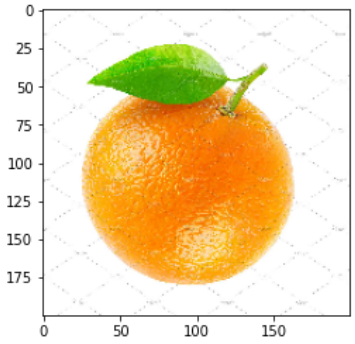
orange_89.jpg is orange



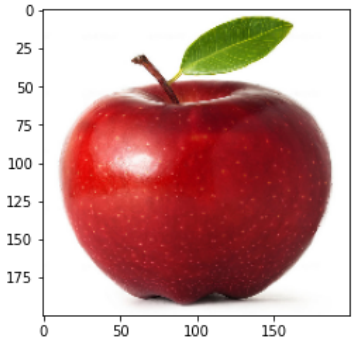
orange_82.jpg is orange



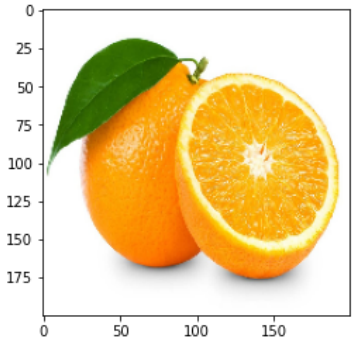
orange_84.jpg is orange



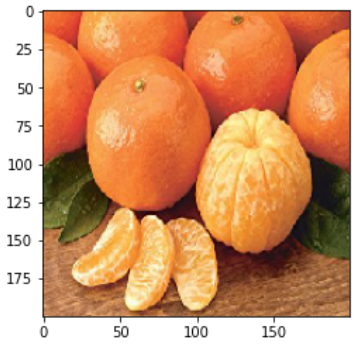
orange_95.jpg is orange



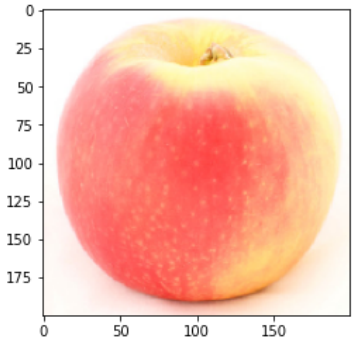
apple_81.jpg is apple



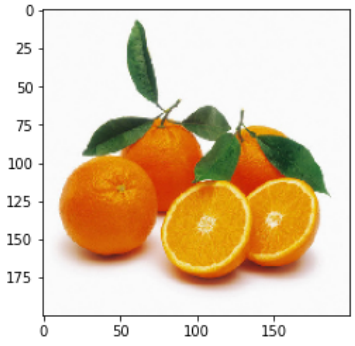
orange_80.jpg is orange



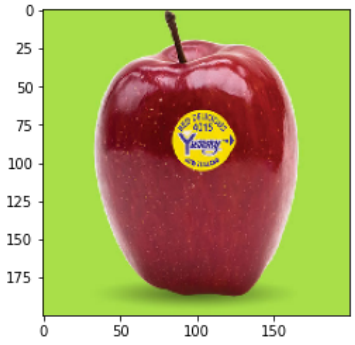
orange_90.jpg is orange



apple_78.jpg is apple

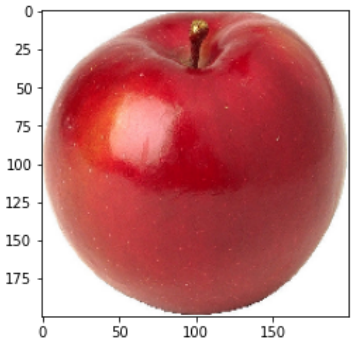


orange_87.jpg is orange

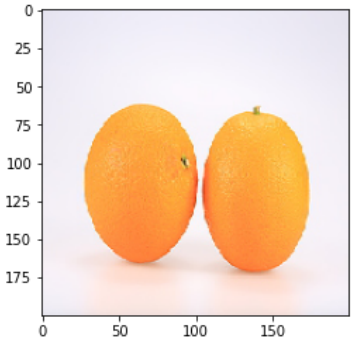


apple_85.jpg is apple

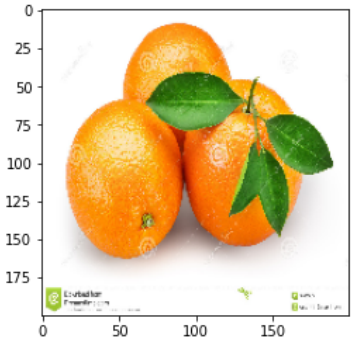
apple_94.jpg is apple



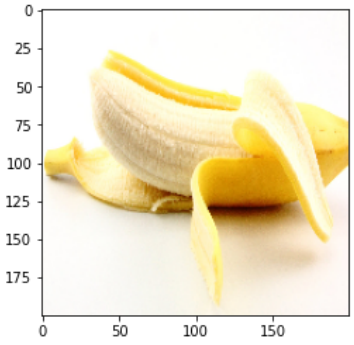
apple_83.jpg is apple



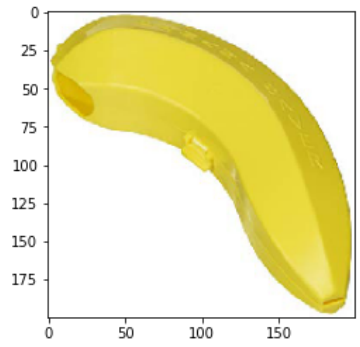
orange_93.jpg is orange



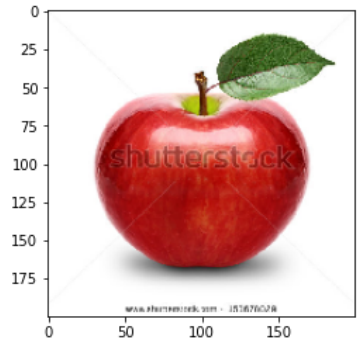
orange_81.jpg is orange



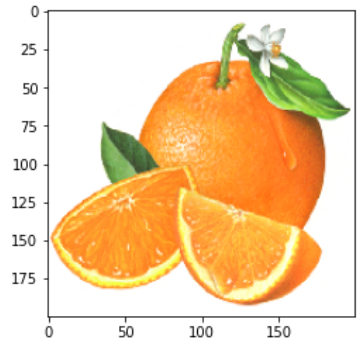
banana_85.jpg is banana



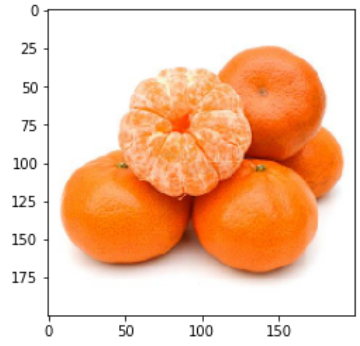
banana_94.jpg is banana



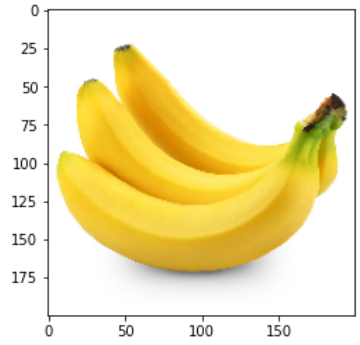
apple_82.jpg is apple



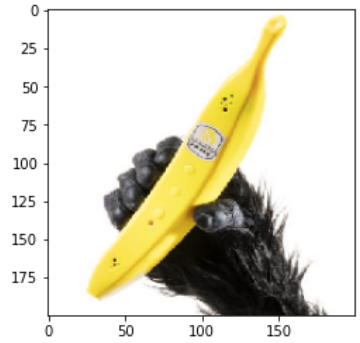
orange_94.jpg is orange



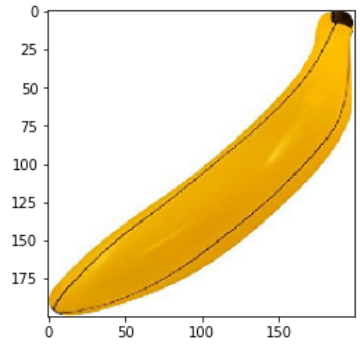
orange_79.jpg is orange



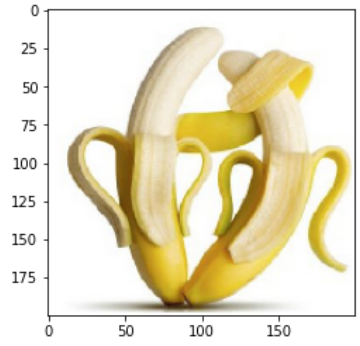
banana_89.jpg is banana



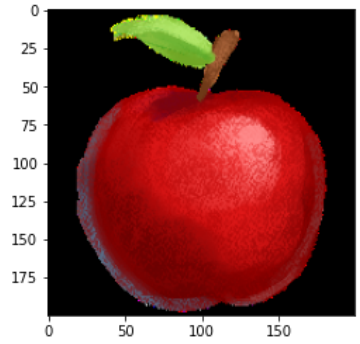
```
banana_77.jpg is banana
```



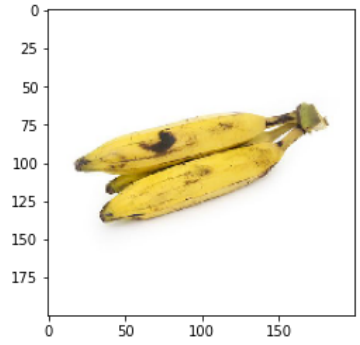
```
banana_92.jpg is banana
```



```
banana_78.jpg is banana
```



```
apple_79.jpg is apple
```



```
banana_80.jpg is banana
Accuracy of test set is 0.945
```