Abstract

This case study is deals with fruit image classification. Dataset is provided for Multi-Class, Single image classification A CNN model is being here designed to based on a size of input. Model is also being trained with ten epochs. Training loss/validation loss are also being presented here. At the end, model is being evaluated with appropriate metrics.

Table of Contents

TITLE

Fruit Classification

1. Introduction

Dataset in this case study is about fruit image classification. The provided dataset is for Multi-Class, Single image classification. It includes training images from several classes and images that model should predict. The images depict a variety of fruits and are organized into five categories: Banana (1439 images), Cherry (1229 images), Grape (1475 images), Mango (915 images), and Peach (1229 images). These images will be trained with a help of sequential model in order to extract best feature from image. Image classification is the process of a computer analyzing an image and determining which 'class' it belongs to.

1.1. Aims

To enhance the accuracy of a model in order to recognize the object from images is the main aim of this classification task.

1.2. Objective

- Reduce the dimension of the images to get the most information and characteristics.
- To map output classes, a fully connected layer flatten output from convolutional layers.
- To identify hyperparameters for sequential model to increase performance.

1.3. CNN and Image classification

A convolutional neural network (CNN) is a form of artificial neural network which is specifically intended to process pixel input and is used in image recognition and processing. CNNs are image processing AI systems that employ deep learning to accomplish both generative and descriptive tasks. It uses computer vision that comes with Image and video recognition, as well as recommender systems and natural language processing.

Because of its great accuracy, CNNs are employed for picture classification and identification. CNNs are feed forward neural networks that are hyperlinked. CNNs are exceptionally good at lowering the number of parameters without sacrificing model quality.

1.4. Outline of a Report

This report aims to study dataset about image classification specially fruit classification. It has a model with its various layers and hyperparameters. Model is being trained and findings are discussed here, which is briefly described in upcoming sections of a report.

2. Methodology

To begin, we divide our data into two sets: one for training and the other for validation. To load our data, we utilize the keras utils library image dataset from the directory, with validation 20% picture size of given height and width and batch size of 32.

2.1. Model summary

In [16]:
```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

```
In [17]: model.compile(optimizer='adam',
                       loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                       metrics=['accuracy'])

In [18]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2D  (None, 90, 90, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
```

i. There are 3 layers; convolution blocks (Conv2D), three max pooling layers (MaxPooling2D), and a fully linked layer (Dense).

ii. Trainable hypermeters are learning rate, batch size, shuffle data while training, optimizer selection, loss function selection.

Conv2D is a crucial component of CNN because it is here that the convolution takes place, i.e., where we stride and multiply our input picture with the filter/kernel. MaxPooling2D adds a filter to a feature map in order to minimize the number of dimensions while maintaining the most important feature information. The Dense Layer is a basic layer of neurons that receives input from all neurons in the previous layer, thus the name. The Dense Layer is used to identify pictures using the convolutional layer's output.

2.2. Training the model

2.2.1. Loss Function

The loss function turns neural networks from glorified matrix multiplication to deep learning and brings algorithm from theoretical to practical. It's a way of determining how effectively algorithm models the data. During the model compilation phase, Keras requires a loss function. In order to train a model, Sparse Categorical Cross entropy loss function is used. The loss in cross-entropy between labels and predictions is calculated. This cross-entropy loss function is used when dealing with multi-class picture categorization.

2.2.2. Optimization Algorithm

The Adam optimization approach is a stochastic gradient descent variant that has lately gained traction in computer vision and natural language processing applications. It repeatedly adjusts the network weights based on training data by comparing the prediction and loss function. Adam is referring to adaptive moment estimation.

2.2.3. Number of Iterations (epochs)

After 10 cycles of training, we evaluate our model's performance. The number of epochs is a hyperparameter that determines how often the learning algorithm runs across the whole training dataset. Every epoch, each sample in the training dataset has the chance to alter the internal model parameters.

Extract the image of training and validation loss against iteration and put it here.

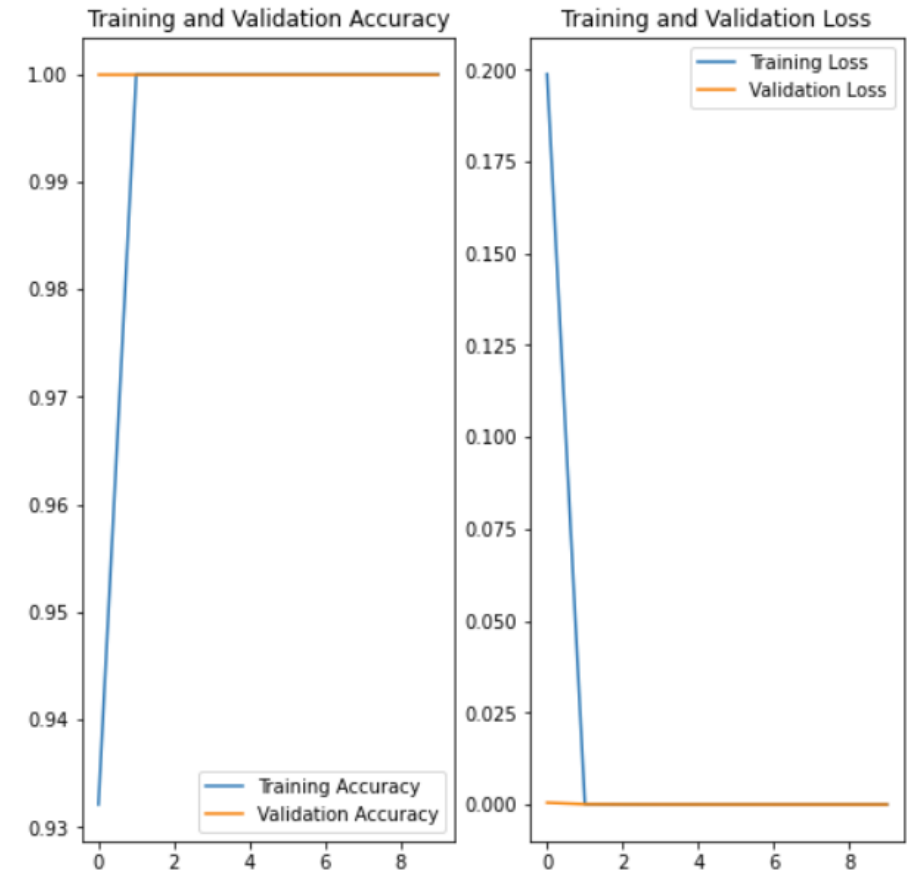
In [20]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']


loss = history.history['loss']
val_loss = history.history['val_loss']


epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



2.3. Findings and Discussion

6. Results and Prediction

In [35]:
```
banana_path = "./Test/1_100(2).jpg"

img = tf.keras.utils.load_img(
    banana_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Identified to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
PIL.Image.open(str(banana_path))
```

```
Identified to Banana with a 100.00 percent confidence.
```

Out[35]:

In [36]:
```
cherry_path = "./Test/5_100.jpg"

img = tf.keras.utils.load_img(
    cherry_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Identified to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
PIL.Image.open(str(cherry_path))
```

```
Identified to Cherry with a 100.00 percent confidence.
```

Out[36]:

In [37]:
```
mango_path = "./Test/11_100.jpg"

img = tf.keras.utils.load_img(
    mango_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Identified to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
PIL.Image.open(str(mango_path))
```

```
Identified to Mango with a 100.00 percent confidence.
```

Out[37]:

In [38]:
```
grape_path = "./Test/14_100.jpg"

img = tf.keras.utils.load_img(
    grape_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Identified to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
PIL.Image.open(str(grape_path))
```

```
Identified to Grape with a 100.00 percent confidence.
```

Out[38]:

In [39]:
```
peach_path = "./Test/26_100.jpg"

img = tf.keras.utils.load_img(
    peach_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Identified to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
PIL.Image.open(str(peach_path))
```

```
Identified to Peach with a 100.00 percent confidence.
```

Out[39]: