Fruit Recognition with CNN

Import libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import os
import matplotlib.image as mpimg
import random
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import load_img, img_to_array
import pathlib
```

```
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:98: UserWarning: unable to load libtensorf
low_io_plugins.so: unable to open file: libtensorflow_io_plugins.so, from paths: ['/opt/conda/lib/python3.10/site-packa
ges/tensorflow_io/python/ops/libtensorflow_io_plugins.so']
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so: undefined sy
mbol: _ZN3tsl6StatusC1EN10tensorflow5error4CodeESt17basic_string_viewIcSt11char_traitsIcEENS_14SourceLocationE']
  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:104: UserWarning: file system plugins are
not loaded: unable to open file: libtensorflow_io.so, from paths: ['/opt/conda/lib/python3.10/site-packages/tensorflow_
io/python/ops/libtensorflow_io.so']
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io.so: undefined symbol: _Z
TVN10tensorflow13GcsFileSystemE']
  warnings.warn(f"file system plugins are not loaded: {e}")
```

In [2]:
```
for dirpath, dirnames, filenames in os.walk("../input/fruit-recognition"):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```
There are 2 directories and 1 images in '../input/fruit-recognition'.
There are 1 directories and 0 images in '../input/fruit-recognition/test'.
There are 0 directories and 5641 images in '../input/fruit-recognition/test/test'.
There are 1 directories and 0 images in '../input/fruit-recognition/train'.
There are 33 directories and 0 images in '../input/fruit-recognition/train/train'.
There are 0 directories and 479 images in '../input/fruit-recognition/train/train/Orange'.
There are 0 directories and 738 images in '../input/fruit-recognition/train/train/Tomato'.
There are 0 directories and 490 images in '../input/fruit-recognition/train/train/Passion Fruit'.
There are 0 directories and 392 images in '../input/fruit-recognition/train/train/Cucumber Ripe'.
There are 0 directories and 490 images in '../input/fruit-recognition/train/train/Cactus fruit'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Pomegranate'.
There are 0 directories and 447 images in '../input/fruit-recognition/train/train/Plum'.
There are 0 directories and 490 images in '../input/fruit-recognition/train/train/Pineapple'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Papaya'.
There are 0 directories and 450 images in '../input/fruit-recognition/train/train/Potato Red'.
There are 0 directories and 466 images in '../input/fruit-recognition/train/train/Kiwi'.
There are 0 directories and 490 images in '../input/fruit-recognition/train/train/Limes'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Apple Braeburn'.
There are 0 directories and 696 images in '../input/fruit-recognition/train/train/Pear'.
There are 0 directories and 438 images in '../input/fruit-recognition/train/train/Onion White'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Strawberry'.
There are 0 directories and 984 images in '../input/fruit-recognition/train/train/Grape Blue'.
There are 0 directories and 462 images in '../input/fruit-recognition/train/train/Blueberry'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Apple Granny Smith'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Apricot'.
There are 0 directories and 666 images in '../input/fruit-recognition/train/train/Pepper Red'.
There are 0 directories and 490 images in '../input/fruit-recognition/train/train/Clementine'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Lemon'.
There are 0 directories and 427 images in '../input/fruit-recognition/train/train/Avocado'.
There are 0 directories and 490 images in '../input/fruit-recognition/train/train/Raspberry'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Cantaloupe'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Peach'.
There are 0 directories and 450 images in '../input/fruit-recognition/train/train/Corn'.
There are 0 directories and 490 images in '../input/fruit-recognition/train/train/Banana'.
There are 0 directories and 492 images in '../input/fruit-recognition/train/train/Cherry'.
There are 0 directories and 444 images in '../input/fruit-recognition/train/train/Pepper Green'.
There are 0 directories and 475 images in '../input/fruit-recognition/train/train/Watermelon'.
There are 0 directories and 490 images in '../input/fruit-recognition/train/train/Mango'.
```

Setting path for TRAIN AND TEST

In [3]:
```
train_path = "../input/fruit-recognition/train/train/"
test_path = "../input/fruit-recognition/test/test/"
```

Getting the Different class names from the directory

In [4]:
```
data_dir = pathlib.Path(train_path)
class_names = np.array(sorted([item.name for item in data_dir.glob('*')]))
print(class_names)
```

```
['Apple Braeburn' 'Apple Granny Smith' 'Apricot' 'Avocado' 'Banana'
 'Blueberry' 'Cactus fruit' 'Cantaloupe' 'Cherry' 'Clementine' 'Corn'
 'Cucumber Ripe' 'Grape Blue' 'Kiwi' 'Lemon' 'Limes' 'Mango' 'Onion White'
 'Orange' 'Papaya' 'Passion Fruit' 'Peach' 'Pear' 'Pepper Green'
 'Pepper Red' 'Pineapple' 'Plum' 'Pomegranate' 'Potato Red' 'Raspberry'
 'Strawberry' 'Tomato' 'Watermelon']
```

Plot a grid of images from a training dataset

In [5]:
```
def view_random_image(target_dir, target_class):
  # Setup target directory (we'll view images from here)
  target_folder = target_dir+target_class

  # Get a random image path
  random_image = random.sample(os.listdir(target_folder), 100)

  # Read in the image and plot it using matplotlib
  img = mpimg.imread(target_folder + "/" + random_image[0])
  plt.imshow(img)
  plt.title(target_class)
  plt.axis("off");

  return img
```

In [6]:
```
plt.figure(figsize = (15,15))
# View a random image from the training dataset for all classes
for i in range(33):
    plt.subplot(5,8,i+1)
    img = view_random_image(target_dir=train_path,
                            target_class=class_names[i])
```


Prepare the VALIDATION Data

Collects Information about the training dataset by traversing the directory tree using os.walk() and populating a dictionary called train_val_data.

In [7]:
```
train_val_data = {'path' : [],
       'filename': [],
       'label': []}
for dirpath, dirnames, filenames in os.walk(train_path):
    for f in filenames:
```

```
        train_val_data['path'].append(dirpath)
        train_val_data['filename'].append(f)
        train_val_data['label'].append(f.split('_')[0])
```

In [8]:
```
train_val_data_df = pd.DataFrame(train_val_data)
train_val_data_df.head()
```

Out[8]:

	path	filename	label
0	../input/fruit-recognition/train/train/Orange	Orange_398.jpg	Orange
1	../input/fruit-recognition/train/train/Orange	Orange_20.jpg	Orange
2	../input/fruit-recognition/train/train/Orange	Orange_337.jpg	Orange
3	../input/fruit-recognition/train/train/Orange	Orange_190.jpg	Orange
4	../input/fruit-recognition/train/train/Orange	Orange_456.jpg	Orange

Prepare the Testing Data

In [10]:
```
#Read test data and create a dataframe
test_data = {'path' : [],
       'filename': []}
for dirpath, dirnames, filenames in os.walk(test_path):
    for f in filenames:
        test_data['path'].append(dirpath)
        test_data['filename'].append(f)
```

In [11]:
```
test_data_df = pd.DataFrame(test_data)
test_data_df.head()
```

Out[11]:

	path	filename
0	../input/fruit-recognition/test/test/	0664.jpg
1	../input/fruit-recognition/test/test/	1269.jpg
2	../input/fruit-recognition/test/test/	3863.jpg
3	../input/fruit-recognition/test/test/	2193.jpg
4	../input/fruit-recognition/test/test/	0733.jpg

Read Images and create numpy data array

In [12]:
```
images = []
label = []

for _, d in train_val_data_df.iterrows():
    img = load_img(os.path.join(d['path'],d['filename']))
    images.append(img_to_array(img))
    label.append(d['label'])
```

In [13]:
```
images = np.array(images)
labels = np.array(label)
print(f"Complete data images shape: {images.shape} and label shape: {labels.shape}")
```

```
Complete data images shape: (16854, 100, 100, 3) and label shape: (16854,)
```

In [14]:
```
test_images = []

for _, d in test_data_df.iterrows():
    img = load_img(os.path.join(d['path'],d['filename']))
    test_images.append(img_to_array(img))

test_images = np.array(test_images)
print(f"Test images shape: {test_images.shape} ")
```

```
Test images shape: (5641, 100, 100, 3)
```

Label Encoding

In [15]:
```
class_indices = dict(zip(class_names, range(len(class_names))))

labels_encoded = list(map(class_indices.get, labels))

#Convert to categorical data using tensorflow
#labels to One-hot encoded
label_categorical = tf.keras.utils.to_categorical(labels_encoded, num_classes=len(class_names), dtype='uint8')
```

Train And Validation split

In [16]:
```
train_im, valid_im, train_lab, valid_lab = train_test_split(images, label_categorical, test_size=0.20,
                                                            stratify=label_categorical,
                                                            random_state=40, shuffle = True)
```

In [17]:
```
print ("train data shape after the split: ", train_im.shape)
print ('new validation data shape: ', valid_im.shape)
print ("validation labels shape: ", valid_lab.shape)
```

```
train data shape after the split:  (13483, 100, 100, 3)
new validation data shape:  (3371, 100, 100, 3)
validation labels shape:  (3371, 33)
```

In [18]:
```
print ('train im and label types: ', type(train_im), type(train_lab))

training_data = tf.data.Dataset.from_tensor_slices((train_im, train_lab))
validation_data = tf.data.Dataset.from_tensor_slices((valid_im, valid_lab))
test_data = tf.data.Dataset.from_tensor_slices(test_images)

print ('check types; ', type(training_data), type(validation_data), type(test_data))
```

```
train im and label types:  <class 'numpy.ndarray'> <class 'numpy.ndarray'>
check types;  <class 'tensorflow.python.data.ops.from_tensor_slices_op._TensorSliceDataset'> <class 'tensorflow.python.
data.ops.from_tensor_slices_op._TensorSliceDataset'> <class 'tensorflow.python.data.ops.from_tensor_slices_op._TensorSl
iceDataset'>
```

In [19]:
```
### check using element_spec

print (training_data.element_spec)
print (validation_data.element_spec)

### as expected, tensors of image and original label shape


### create an iterator and turn it into numpy array
train_iter = iter(training_data)
print(next(train_iter)[0].numpy(), '\n', next(train_iter)[1].numpy(), np.argmax(next(train_iter)[1].numpy()))
```

```
(TensorSpec(shape=(100, 100, 3), dtype=tf.float32, name=None), TensorSpec(shape=(33,), dtype=tf.uint8, name=None))
(TensorSpec(shape=(100, 100, 3), dtype=tf.float32, name=None), TensorSpec(shape=(33,), dtype=tf.uint8, name=None))
[[[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.]]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0] 3
```

In [20]:
```
train_iter_im, train_iter_label = next(iter(training_data))
print (train_iter_im.numpy().shape, train_iter_label.numpy().shape)
```

```
(100, 100, 3) (33,)
```

In [21]:
```
train_iter_im1, train_iter_label1 = next(training_data.as_numpy_iterator())
print (train_iter_im1.shape, train_iter_label1.shape)
```

```
(100, 100, 3) (33,)
```

In [22]:
```
check_list = list(training_data.as_numpy_iterator())
print (len(check_list), check_list[1])
```

```
13483 (array([[[254., 255., 255.],
        [254., 255., 255.],
        [255., 255., 255.],
        ...,
        [255., 255., 255.],
        [255., 255., 255.],
        [255., 255., 255.]],

       [[254., 255., 255.],
        [254., 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.]]], dtype=float32), array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=uint8))
```

In [23]:

```
fig = plt.figure(figsize=(10,10))
for i in range(12):
    plt.subplot(4,3,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(check_list[i][0]/255.)
    plt.xlabel(class_names[np.argmax(check_list[i][1])], fontsize=13)
plt.tight_layout()
plt.show()
```

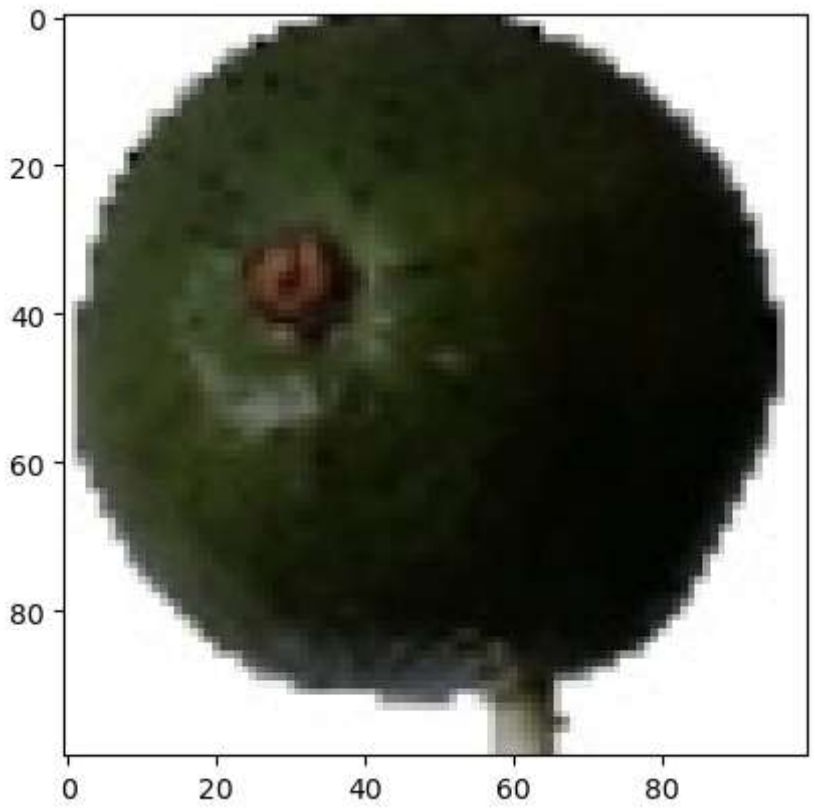
Data Pipeline using `tf.data` & Prefetching

In [24]:
```
rescale_data = tf.keras.Sequential([
    layers.experimental.preprocessing.Rescaling(1/255.)
])

data_augmenation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip(mode = "horizontal"),
    #layers.experimental.preprocessing.RandomRotation(0.1)
])
```

In [25]:
```
random_image_index = random.randint(0,len(train_im))
img = rescale_data(train_im[random_image_index])
img = data_augmenation(img)
plt.imshow(img)
```

Out[25]:
```
<matplotlib.image.AxesImage at 0x7b29960c00d0>
```



In [26]:
```
BATCH_SIZE = 128
AUTOTUNE = tf.data.AUTOTUNE

def prepare(ds, shuffle=False, augment = False, test = False):
    if test:
        ds = ds.map(lambda x: (rescale_data(x)), num_parallel_calls=AUTOTUNE)
    else:
        ds = ds.map(lambda x, y: (rescale_data(x), y), num_parallel_calls=AUTOTUNE)

    if shuffle:
        ds = ds.shuffle(1000)

    #batch the data
    ds = ds.batch(BATCH_SIZE)

    # Use data augmentation only on the training set.
    if augment:
        ds = ds.map(lambda x, y: (data_augmenation(x, training=True), y),
                num_parallel_calls=AUTOTUNE)

    # Use buffered prefetching on all datasets.
    return ds.prefetch(buffer_size=AUTOTUNE)
```

In [27]:
```
train_ds = prepare(training_data, shuffle = True, augment = True)
val_ds = prepare(validation_data)
test_ds = prepare(test_data, test=True)
```

CNN Model

- *The model consists of a convolutional layer, a max pooling layer, a flatten layer, and a dense (fully connected) layer with a softmax activation function*

In [28]:
```
model_1 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=5,
                           kernel_size = 3,
                           activation = "relu",
                           input_shape = (100,100,3)),
    tf.keras.layers.MaxPool2D(pool_size =2,
                              padding='valid'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(len(class_names), activation="softmax")
])

model_1.compile(loss="categorical_crossentropy",
                optimizer = tf.keras.optimizers.Adam(),
                metrics = ['accuracy'])
```

In [29]:
```
model_1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 5)         140

 max_pooling2d (MaxPooling2D  (None, 49, 49, 5)        0
 )

 flatten (Flatten)           (None, 12005)             0

 dense (Dense)               (None, 33)                396198

=================================================================
Total params: 396,338
Trainable params: 396,338
Non-trainable params: 0
_________________________________________________________________
```

Fitting The Model

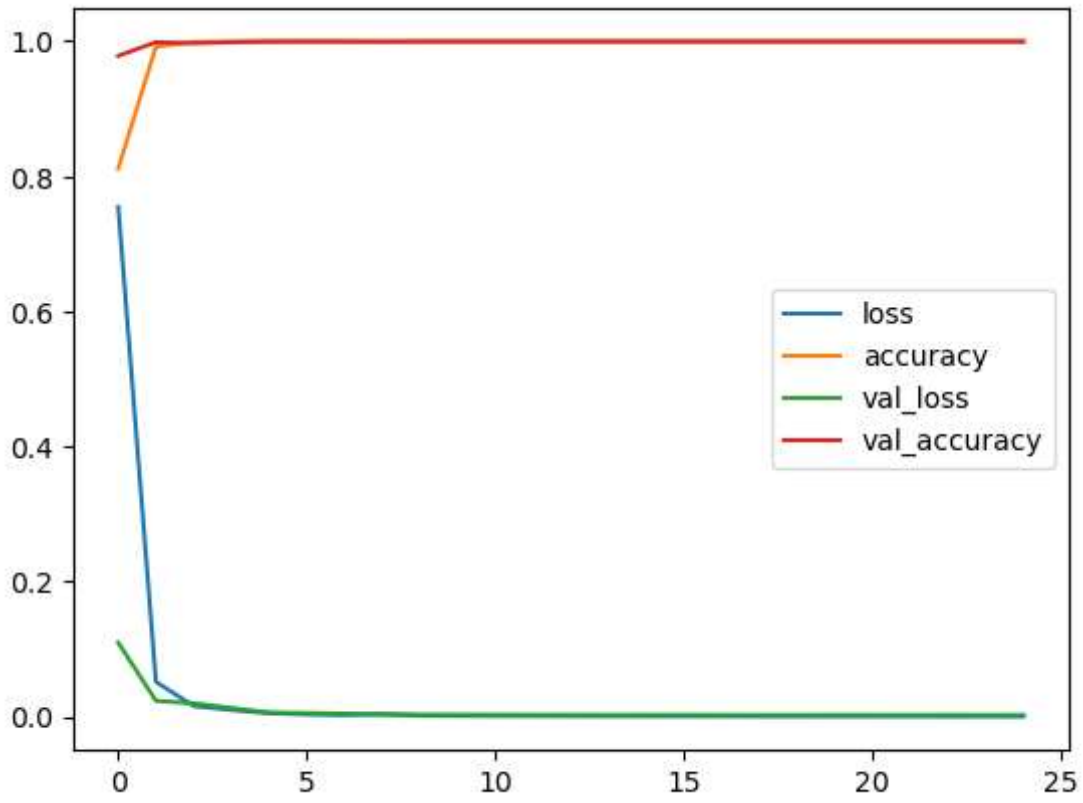
In [30]:
```
#Fit the model on training data
history_1 = model_1.fit(train_ds,
                        epochs =25,
                        validation_data= val_ds)
```

```
Epoch 1/25
106/106 [==============================] - 13s 42ms/step - loss: 0.7546 - accuracy: 0.8114 - val_loss: 0.1095 - val_acc
uracy: 0.9783
Epoch 2/25
106/106 [==============================] - 4s 35ms/step - loss: 0.0514 - accuracy: 0.9919 - val_loss: 0.0233 - val_accu
racy: 0.9985
Epoch 3/25
106/106 [==============================] - 4s 37ms/step - loss: 0.0157 - accuracy: 0.9990 - val_loss: 0.0197 - val_accu
racy: 0.9970
Epoch 4/25
106/106 [==============================] - 4s 32ms/step - loss: 0.0098 - accuracy: 0.9996 - val_loss: 0.0126 - val_accu
racy: 0.9985
Epoch 5/25
106/106 [==============================] - 4s 34ms/step - loss: 0.0049 - accuracy: 0.9999 - val_loss: 0.0060 - val_accu
racy: 0.9994
Epoch 6/25
106/106 [==============================] - 4s 35ms/step - loss: 0.0031 - accuracy: 1.0000 - val_loss: 0.0052 - val_accu
racy: 0.9994
Epoch 7/25
106/106 [==============================] - 4s 36ms/step - loss: 0.0023 - accuracy: 1.0000 - val_loss: 0.0044 - val_accu
racy: 0.9994
Epoch 8/25
106/106 [==============================] - 4s 35ms/step - loss: 0.0036 - accuracy: 0.9994 - val_loss: 0.0032 - val_accu
racy: 0.9994
Epoch 9/25
106/106 [==============================] - 5s 45ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.0029 - val_accu
racy: 0.9994
Epoch 10/25
106/106 [==============================] - 3s 31ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.0028 - val_accu
racy: 0.9994
Epoch 11/25
106/106 [==============================] - 4s 35ms/step - loss: 8.7904e-04 - accuracy: 1.0000 - val_loss: 0.0027 - val_
accuracy: 0.9994
Epoch 12/25
106/106 [==============================] - 5s 41ms/step - loss: 7.5534e-04 - accuracy: 1.0000 - val_loss: 0.0025 - val_
accuracy: 0.9994
Epoch 13/25
106/106 [==============================] - 4s 36ms/step - loss: 6.3524e-04 - accuracy: 1.0000 - val_loss: 0.0026 - val_
accuracy: 0.9994
Epoch 14/25
106/106 [==============================] - 4s 34ms/step - loss: 5.5797e-04 - accuracy: 1.0000 - val_loss: 0.0024 - val_
accuracy: 0.9994
Epoch 15/25
106/106 [==============================] - 4s 33ms/step - loss: 4.7677e-04 - accuracy: 1.0000 - val_loss: 0.0024 - val_
accuracy: 0.9994
Epoch 16/25
106/106 [==============================] - 4s 32ms/step - loss: 4.2308e-04 - accuracy: 1.0000 - val_loss: 0.0024 - val_
accuracy: 0.9994
Epoch 17/25
106/106 [==============================] - 5s 43ms/step - loss: 3.8190e-04 - accuracy: 1.0000 - val_loss: 0.0023 - val_
accuracy: 0.9994
Epoch 18/25
106/106 [==============================] - 4s 33ms/step - loss: 3.3268e-04 - accuracy: 1.0000 - val_loss: 0.0023 - val_
accuracy: 0.9994
Epoch 19/25
106/106 [==============================] - 4s 32ms/step - loss: 2.9886e-04 - accuracy: 1.0000 - val_loss: 0.0023 - val_
accuracy: 0.9994
Epoch 20/25
106/106 [==============================] - 4s 39ms/step - loss: 2.7133e-04 - accuracy: 1.0000 - val_loss: 0.0024 - val_
accuracy: 0.9994
Epoch 21/25
106/106 [==============================] - 3s 32ms/step - loss: 2.3337e-04 - accuracy: 1.0000 - val_loss: 0.0023 - val_
accuracy: 0.9994
Epoch 22/25
106/106 [==============================] - 4s 36ms/step - loss: 2.1171e-04 - accuracy: 1.0000 - val_loss: 0.0023 - val_
accuracy: 0.9994
Epoch 23/25
106/106 [==============================] - 4s 31ms/step - loss: 1.8966e-04 - accuracy: 1.0000 - val_loss: 0.0024 - val_
accuracy: 0.9994
Epoch 24/25
106/106 [==============================] - 4s 37ms/step - loss: 1.7514e-04 - accuracy: 1.0000 - val_loss: 0.0023 - val_
accuracy: 0.9994
Epoch 25/25
106/106 [==============================] - 5s 43ms/step - loss: 1.5733e-04 - accuracy: 1.0000 - val_loss: 0.0023 - val_
accuracy: 0.9994
```

Plotting Graph for loss & accuracy

In [31]:
```
pd.DataFrame(history_1.history).plot()
```

Out[31]:
```
<Axes: >
```



Saving The Model

In [32]:
```
model_1.save('model.h5')
```

In [33]:
```
loaded_model = tf.keras.models.load_model('/working/model.h5')
```