```
cd /content/sample_data/Test
```

```
/content/sample_data/Test
```

```
!kaggle datasets download -d muratkokludataset/rice-image-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset
License(s): CC0-1.0
Downloading rice-image-dataset.zip to /content/sample_data/Test
 96% 211M/219M [00:01<00:00, 163MB/s]
100% 219M/219M [00:01<00:00, 127MB/s]
```

```
!unzip /content/sample_data/Test/rice-image-dataset.zip
```

```
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9983).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9984).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9985).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9986).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9987).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9988).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9989).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (999).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9990).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9991).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9992).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9993).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9994).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9995).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9996).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9997).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9998).jpg
  inflating: Rice_Image_Dataset/Karacadag/Karacadag (9999).jpg
  inflating: Rice_Image_Dataset/Rice_Citation_Request.txt
```

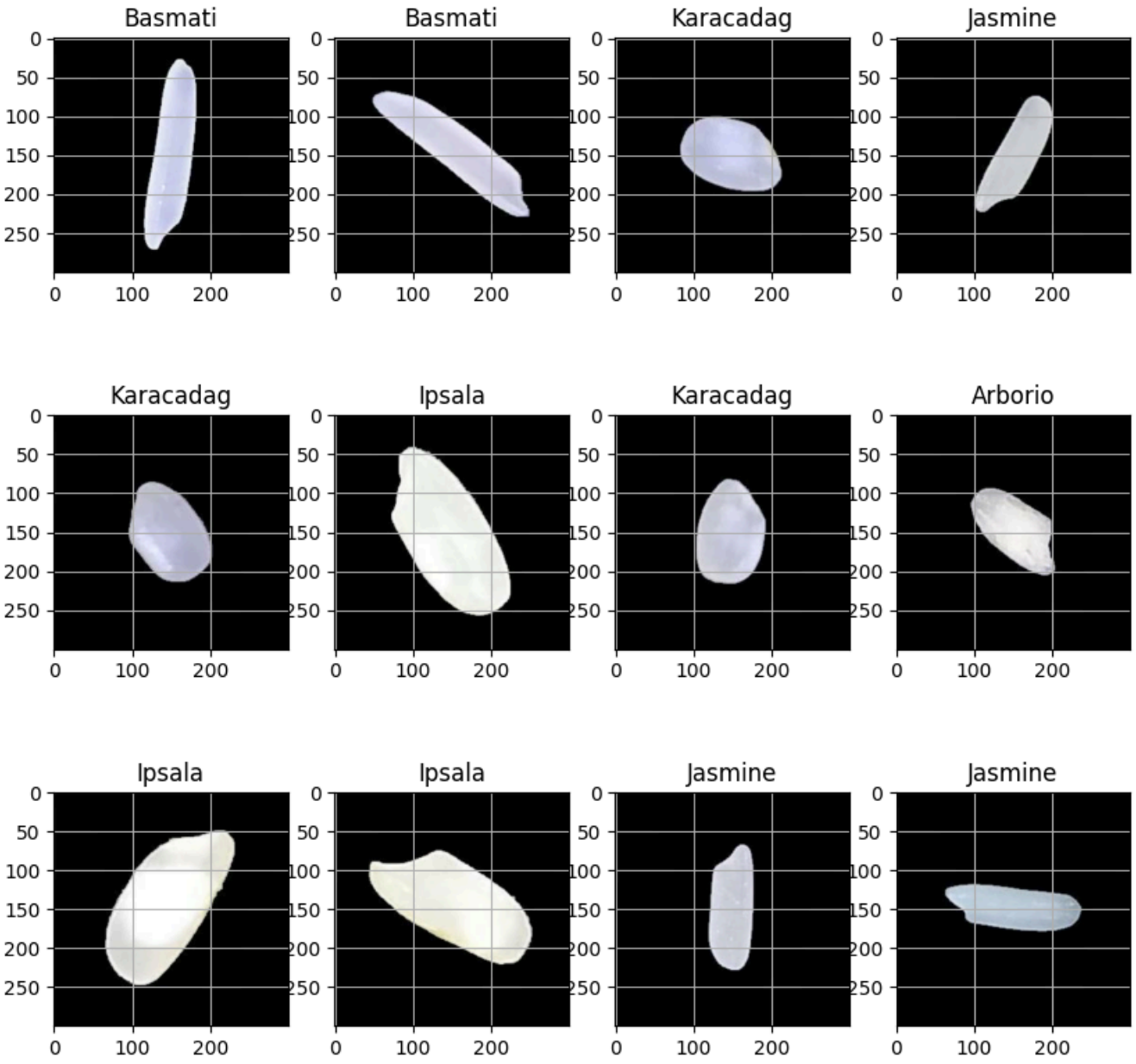
```
## importing essential Libraries

import os
import pandas as pd
import numpy as np
import sys
import seaborn as sb
import tensorflow as tf
from tensorflow.keras import layers, models, backend
import matplotlib.pyplot as plt

## batch specification
batch_size = 50
img_height = 300
img_width = 300


## loading training set
training_ds = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/sample_data/Test/Rice_Image_Dataset',
    validation_split=0.2,
    subset= "training",
    seed=42,
    image_size= (img_height, img_width),
    batch_size=batch_size
)

## loading testing data
testing_ds = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/sample_data/Test/Rice_Image_Dataset',
    validation_split=0.2,
    subset= "validation",
    seed=42,
    image_size= (img_height, img_width),
    batch_size=batch_size
)


class_names = training_ds.class_names
```

```
Found 75000 files belonging to 5 classes.
Using 60000 files for training.
Found 75000 files belonging to 5 classes.
Using 15000 files for validation.
```

```
class_names
```

```
['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
```

```
plt.figure(figsize=(10, 10))
for images, labels in training_ds.take(1):
  for i in range(12):
    ax = plt.subplot(3, 4, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.grid(True)
```



```
model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width,3)),
    layers.Conv2D(16, 3, activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.BatchNormalization(),

    layers.Conv2D(32, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.BatchNormalization(),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),

    layers.Dense(32, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])
```

```
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False, reduction=tf.keras.lo
metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 300, 300, 3)       0

 conv2d_4 (Conv2D)           (None, 298, 298, 16)      448

 max_pooling2d_4 (MaxPoolin  (None, 149, 149, 16)      0
 g2D)

 batch_normalization_2 (Bat  (None, 149, 149, 16)      64
 chNormalization)

 conv2d_5 (Conv2D)           (None, 147, 147, 32)      4640

 max_pooling2d_5 (MaxPoolin  (None, 73, 73, 32)        0
 g2D)

 batch_normalization_3 (Bat  (None, 73, 73, 32)        128
 chNormalization)

 flatten_2 (Flatten)         (None, 170528)            0

 dense_4 (Dense)             (None, 64)                10913856

 dropout (Dropout)           (None, 64)                0
```

```
 dense_5 (Dense)             (None, 32)                2080

 dense_6 (Dense)             (None, 5)                 165

=================================================================
Total params: 10921381 (41.66 MB)
Trainable params: 10921285 (41.66 MB)
Non-trainable params: 96 (384.00 Byte)
_________________________________________________________________
```

```
epochs = 10
history = model.fit(
training_ds,
validation_data=testing_ds,
epochs=epochs
)
```

```
Epoch 1/10
1200/1200 [==============================] - 166s 138ms/step - loss: 0.6362 - accuracy:
Epoch 2/10
1200/1200 [==============================] - 166s 138ms/step - loss: 0.4692 - accuracy:
Epoch 3/10
1200/1200 [==============================] - 164s 136ms/step - loss: 0.3849 - accuracy:
Epoch 4/10
1200/1200 [==============================] - 121s 100ms/step - loss: 0.3032 - accuracy:
Epoch 5/10
1200/1200 [==============================] - 144s 119ms/step - loss: 0.2594 - accuracy:
Epoch 6/10
1200/1200 [==============================] - 143s 119ms/step - loss: 0.2340 - accuracy:
Epoch 7/10
1200/1200 [==============================] - 128s 106ms/step - loss: 0.1898 - accuracy:
Epoch 8/10
1200/1200 [==============================] - 138s 115ms/step - loss: 0.1588 - accuracy:
Epoch 9/10
1200/1200 [==============================] - 148s 123ms/step - loss: 0.1487 - accuracy:
Epoch 10/10
1200/1200 [==============================] - 144s 120ms/step - loss: 0.1386 - accuracy:
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

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

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

# Display the maximum validation accuracy
print("Maximum Validation Accuracy:", max(val_acc))
```

```
AccuracyVector = []
plt.figure(figsize=(30, 20))
for images, labels in testing_ds.take(1):
    predictions = model.predict(images)
    predlabel = []
    prdlbl = []

    for mem in predictions:
        predlabel.append(class_names[np.argmax(mem)])
        prdlbl.append(np.argmax(mem))

    AccuracyVector = np.array(prdlbl) == labels
    for i in range(20):
        ax = plt.subplot(5, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title('Pred: '+ predlabel[i]+' actl:'+class_names[labels[i]] )
        plt.axis('off')
        plt.grid(True)
```

```
2/2 [==============================] - 1s 266ms/step
```