THE UNIVERSITY OF DODOMA

COLLEGE OF INFORMATICS AND VIRTUAL EDUCATION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ASSIGNMENT 04

CONVOLUTIONAL NEURAL NETWORKS (CNN) REPORT

TITLE: Rice classification dataset

COURSE: BIG DATA

CODE: CP 429

INSTRUCTOR: DR. MURO

GROUP MEMBERS

STUDENT'S NAME	REG. NUMBER
RAMADHANI MUSA MTANI	T/UDOM/2020/00383
RAMADHANI KASIMU BAKARI	T/UDOM/2020/00366
HASSAN Z. MNYAWAN	T/UDOM/2020/00392
HUSSEIN MCHENI HASSANI	T/UDOM/2020/06793
SALOME ELIAS	T/UDOM/2020/00381

Introduction

Rice, one of the most widely produced grain products globally, has numerous genetic varieties. These varieties are distinguished by features such as texture, shape, and color. These distinguishing characteristics allow for the classification and quality assessment of rice seeds.

This dataset has pictures of 5 sorts of rice counting: Arborio, Basmati, Ipsala, Jasmine and Karacadag.

The dataset has 75K pictures counting 15K pieces from each rice assortment. By building a model for this dataset, we will utilize it to classify rice sorts.

Working on Convolutional Neural Networks

Importing library

```
import splitfolders
import itertools
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import confusion_matrix, classification_report
```

Preparing Dataset

```
Data ="Rice_Image_Dataset"
```

```
# splite dataset to 3 class of train, validation and test
splitfolders.ratio(Data, output='Rice_Image_Train_Validation_Test', seed=42, ratio=(.7, .2, .1), group_prefix=None)
```

```
# read train, validation and test in dataset
BATCH_SIZE = 20

IMAGE_SIZE = (160, 160)

Train = keras.utils.image_dataset_from_directory(

    directory = 'Rice_Image_Train_Validation_Test/train',
    labels = 'inferred',
    label_mode = 'int',
    batch_size = BATCH_SIZE,
    image_size = IMAGE_SIZE,
    seed = 42,
)

Validation = keras.utils.image_dataset_from_directory(
    directory = 'Rice_Image_Train_Validation_Test/val',
    labels = 'inferred',
    label_mode = 'int',
    batch_size = BATCH_SIZE,
    image_size = IMAGE_SIZE,
    seed = 42,
)
```

```
Test = keras.utils.image_dataset_from_directory(
    directory = 'Rice_Image_Train_Validation_Test/test',
    labels = 'inferred',
    label_mode = 'int',
    batch_size = BATCH_SIZE,
    image_size = IMAGE_SIZE,
    seed = 42,
)
```

```
Found 52500 files belonging to 5 classes.
Found 15000 files belonging to 5 classes.
Found 7500 files belonging to 5 classes.
```

Checking type of classes

```
# check type of each class
print(f"The Train has {len(Train.class_names)} classes called: {Train.class_names}")
print(f"The Validation has {len(Validation.class_names)} classes called: {Validation.class_names}")
print(f"The Test has {len(Test.class_names)} classes called: {Test.class_names}")
```

```
The Train has 5 classes called: ['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
The Validation has 5 classes called: ['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
The Test has 5 classes called: ['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
```

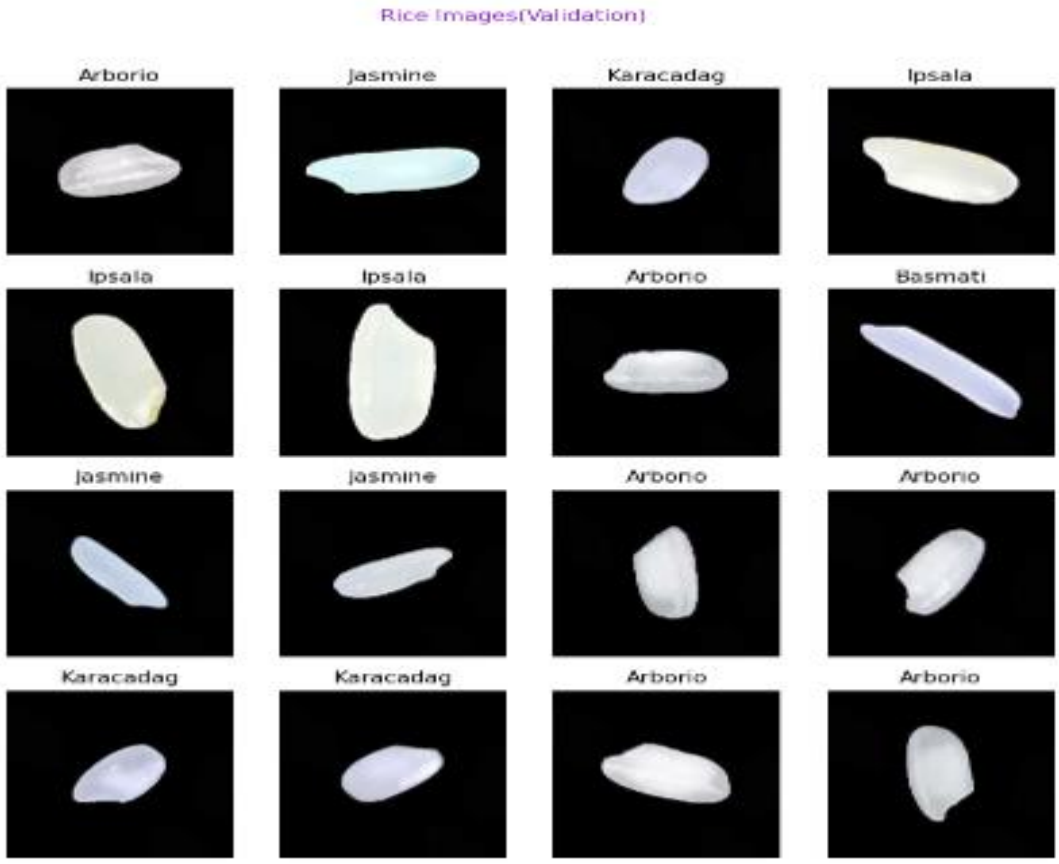
Checking shapes of data

```
# check shapes
for image_batch, labels_batch in Train:
    print(f"Train Shape: {image_batch.shape} (Bathes = {len(Train)})")
    print(f"Train label: {labels_batch.shape}\n")
    break
for image_batch, labels_batch in Validation:
    print(f"Validation Shape: {image_batch.shape} (Bathes = {len(Validation)})")
    print(f"Validation label: {labels_batch.shape}\n")
    break
for image_batch, labels_batch in Test:
    print(f"Test Shape: {image_batch.shape} (Bathes = {len(Test)})")
    print(f"Test label: {labels_batch.shape}\n")
    break
```

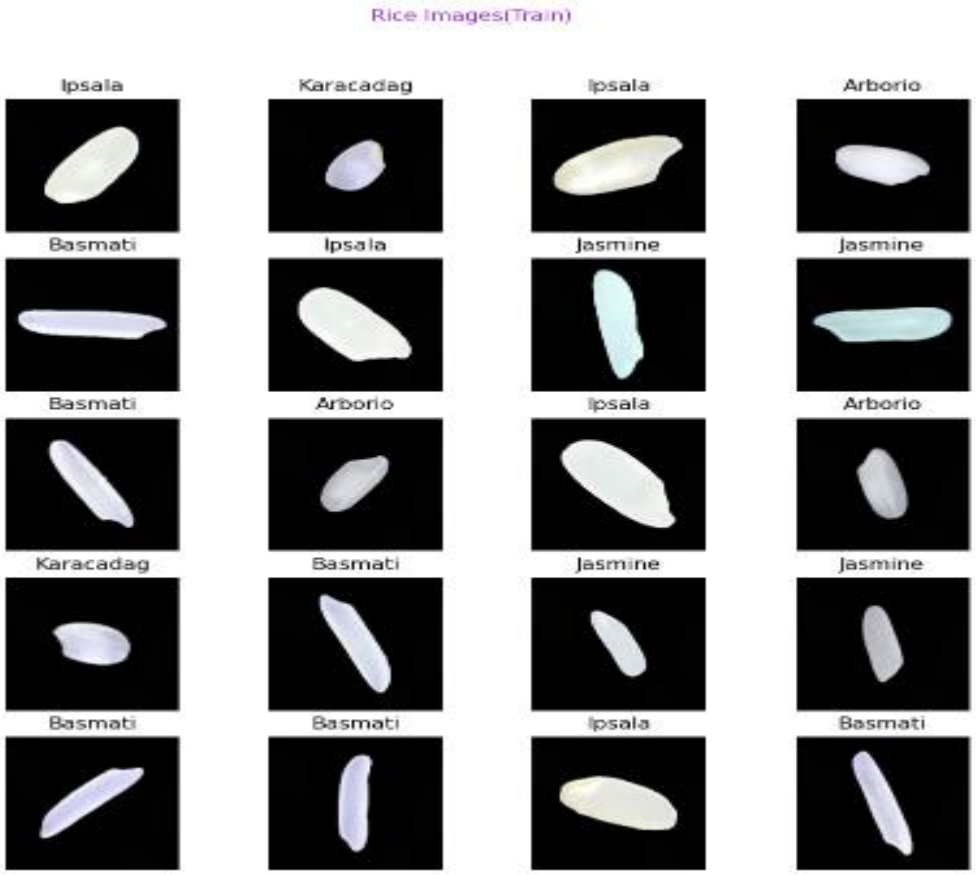
```
Train Shape: (20, 160, 160, 3) (Bathes = 2625)
Train label: (20,)

Validation Shape: (20, 160, 160, 3) (Bathes = 750)
Validation label: (20,)

Test Shape: (20, 160, 160, 3) (Bathes = 375)
Test label: (20,)
```

```
# visualize the Validation to verify is in the correct format

plt.figure(figsize=(10, 10))
for images, labels in Validation.take(1):
  for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(images[i].numpy().astype("uint16"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
    plt.suptitle('Rice Images(Validation)', y=0.96, color='darkorchid')
plt.show()
```

Visualize validation data

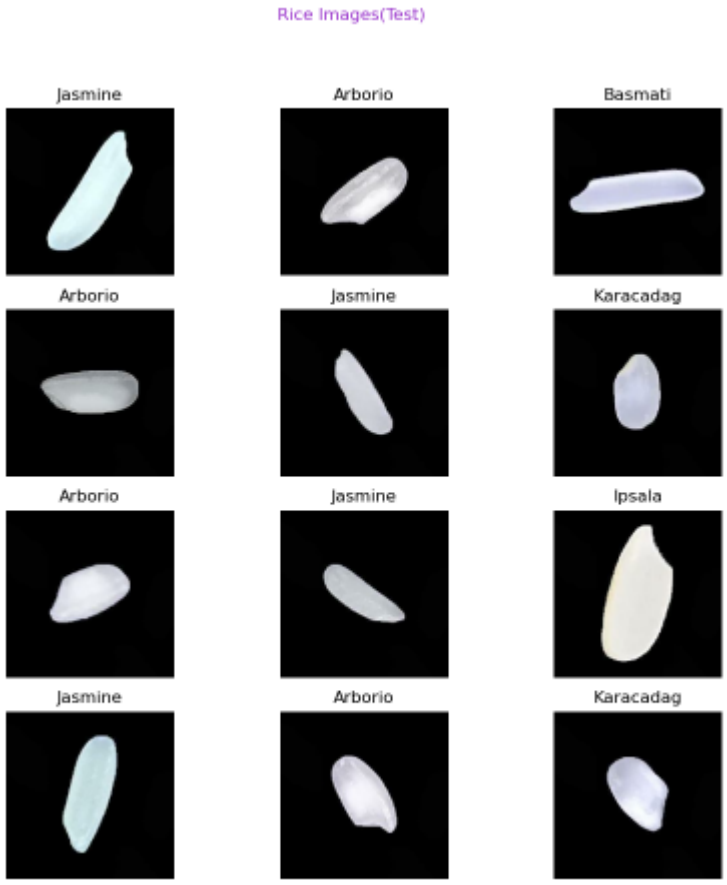


visualize train data



Visualize test data

```
# visualize the Test to verify is in the correct format

plt.figure(figsize=(10, 10))
for images, labels in Test.take(1):
  for i in range(12):
    ax = plt.subplot(4, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
    plt.suptitle('Rice Images(Test)', y=0.98, color='darkorchid')
plt.show()
```



Building model

The followings are pieces of codes snapshots

[13]:
```
# Creat CNN
CNN1 = tf.keras.models.Sequential()
```

[14]:
```
# Convelotion Operation
CNN1.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3,activation='relu', input_shape=(160,160,3)))
```

[15]:
```
# Pooling
CNN1.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2))
```

[16]:
```
# Flattening
CNN1.add(tf.keras.layers.Flatten())
```

[17]:
```
# Full Connection
CNN1.add(tf.keras.layers.Dense(units=320, activation='relu'))
```

[18]:
```
# Output Layers
CNN1.add(tf.keras.layers.Dense(units=5, activation='sigmoid'))
```

[19]:
```
# Training the CNN (set compiler method)
CNN1.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

[20]:
```
# Training the CNN (fit train and validation)

Model1 = CNN1.fit(x=Train, validation_data= Validation, epochs=10)
```

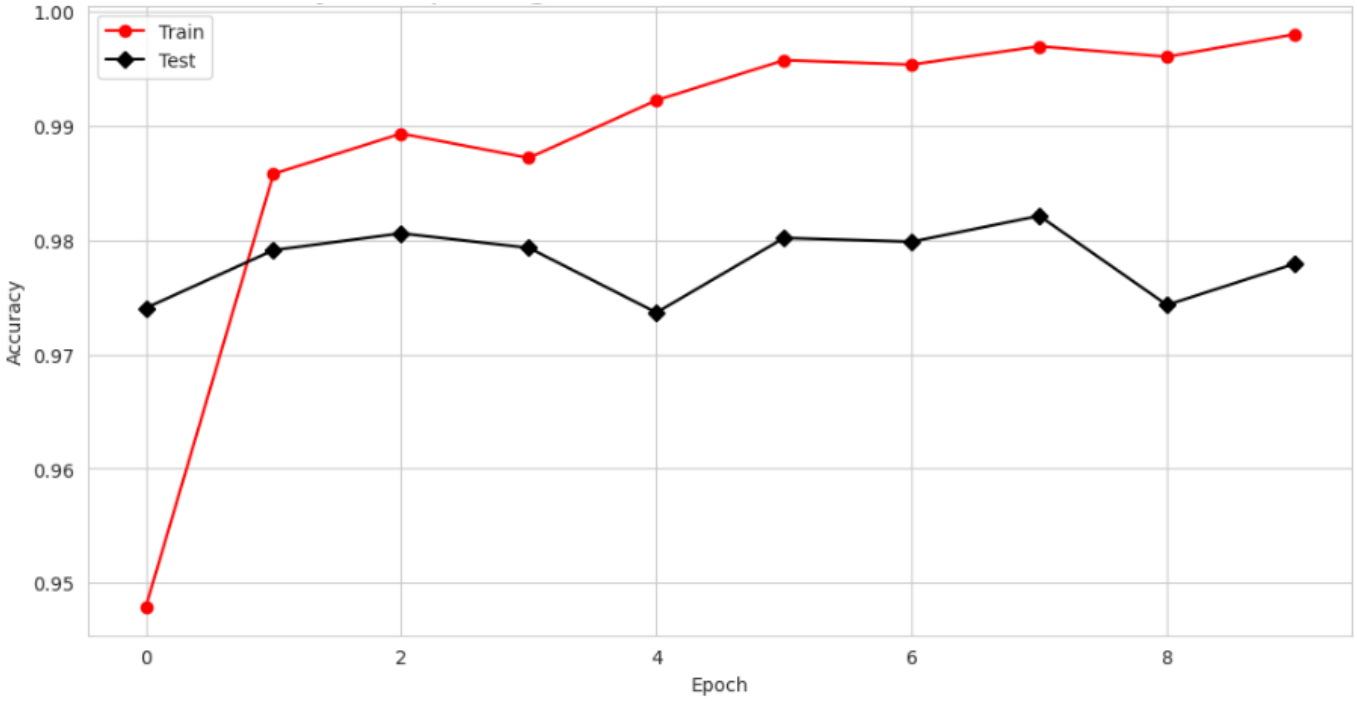
```
Epoch 1/10
2625/2625 [==============================] - 87s 28ms/step - loss: 9.1149 - accuracy: 0.9479 - val_loss: 0.0951 - val_accuracy: 0.9741
Epoch 2/10
2625/2625 [==============================] - 81s 31ms/step - loss: 0.0454 - accuracy: 0.9858 - val_loss: 0.0771 - val_accuracy: 0.9791
Epoch 3/10
2625/2625 [==============================] - 71s 27ms/step - loss: 0.0323 - accuracy: 0.9893 - val_loss: 0.0784 - val_accuracy: 0.9806
Epoch 4/10
2625/2625 [==============================] - 82s 31ms/step - loss: 0.0418 - accuracy: 0.9872 - val_loss: 0.0785 - val_accuracy: 0.9793
Epoch 5/10
2625/2625 [==============================] - 71s 27ms/step - loss: 0.0239 - accuracy: 0.9922 - val_loss: 0.1428 - val_accuracy: 0.9737
Epoch 6/10
2625/2625 [==============================] - 71s 27ms/step - loss: 0.0138 - accuracy: 0.9958 - val_loss: 0.1565 - val_accuracy: 0.9802
Epoch 7/10
2625/2625 [==============================] - 82s 31ms/step - loss: 0.0219 - accuracy: 0.9954 - val_loss: 0.1396 - val_accuracy: 0.9799
Epoch 8/10
2625/2625 [==============================] - 71s 27ms/step - loss: 0.0107 - accuracy: 0.9970 - val_loss: 0.1593 - val_accuracy: 0.9821
Epoch 9/10
2625/2625 [==============================] - 71s 27ms/step - loss: 0.0234 - accuracy: 0.9961 - val_loss: 0.2290 - val_accuracy: 0.9743
Epoch 10/10
2625/2625 [==============================] - 71s 27ms/step - loss: 0.0083 - accuracy: 0.9980 - val_loss: 0.2091 - val_accuracy: 0.9779
```

```
# get summary
CNN1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 158, 158, 32)      896

 max_pooling2d (MaxPooling2D  (None, 79, 79, 32)       0
 )

 flatten (Flatten)           (None, 199712)            0

 dense (Dense)               (None, 320)               63908160

 dense_1 (Dense)             (None, 5)                 1605

=================================================================
Total params: 63,910,661
Trainable params: 63,910,661
Non-trainable params: 0
_________________________________________________________________
```

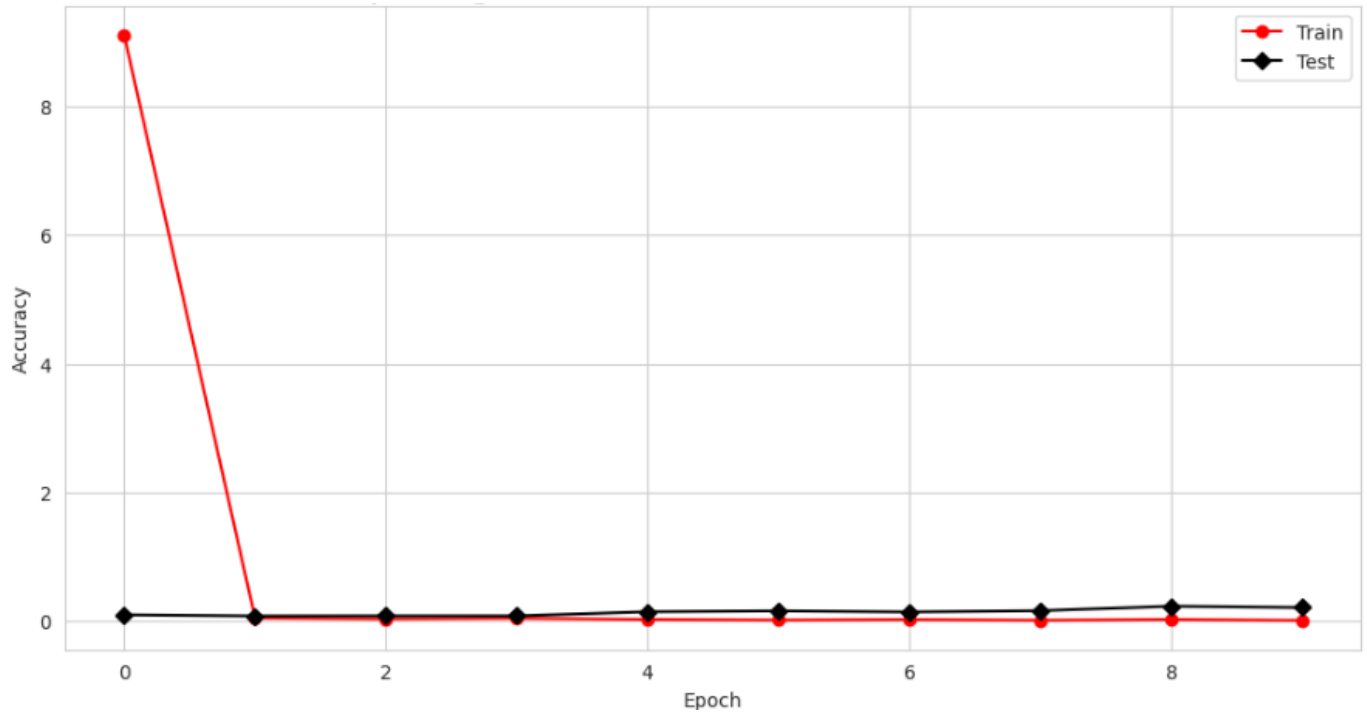
```
plt.figure(figsize=(12,6))
sb.set_style('whitegrid')
plt.plot(Model1.history['accuracy'], color='red', marker="o")
plt.plot(Model1.history['val_accuracy'],color='black', marker="D")
plt.title('Accuracy comparing between Validation and Train data set', fontsize=20, color='black')
plt.xlabel("Epoch")
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
```

Visualization: Accuracy comparing between validation and train dataset



[23]:
```
plt.figure(figsize=(12,6))
sb.set_style('whitegrid')
plt.plot(Model1.history['loss'], color='red', marker="o")
plt.plot(Model1.history['val_loss'],color='black', marker="D")
plt.title('Loss comparing between Validation and Train data set', fontsize=20, color='black')
plt.xlabel("Epoch")
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
```

Visualize: Loss comparing between validation and train dataset

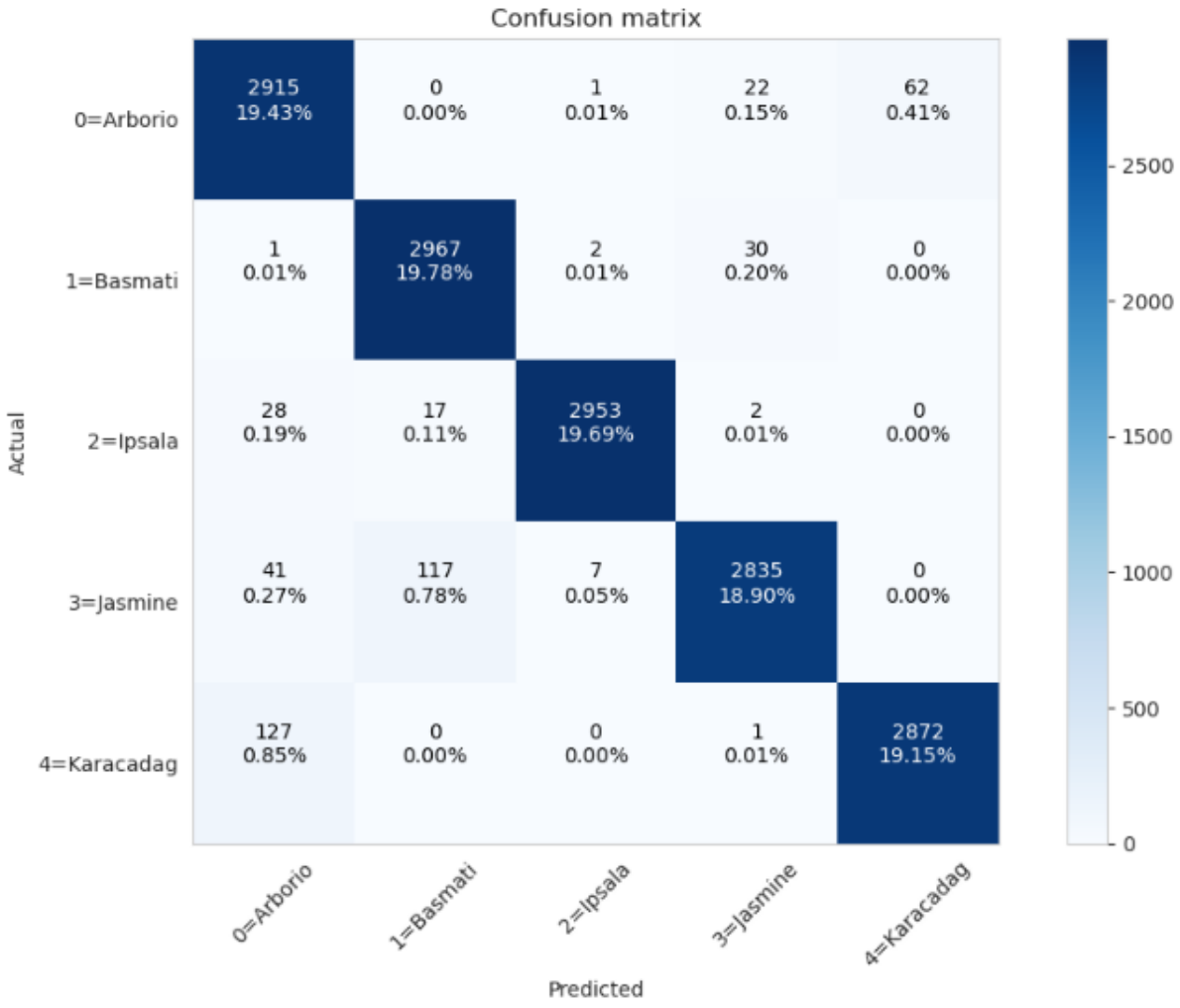


Confusion matrix

[24]:
```
# create a function to draw a confusion matrix
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function plots the confusion matrix.
        cm(array): confusion matrix
        classes(dictionary): classes of our target (key=categorical type, value=numerical type)
    """
    plt.figure(figsize=(10,7))
    plt.grid(False)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))

    plt.xticks(tick_marks, [f"{value}={key}" for key , value in classes.items()], rotation=45)
    plt.yticks(tick_marks, [f"{value}={key}" for key , value in classes.items()])

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, f"{cm[i,j]}\n{cm[i,j]/np.sum(cm)*100:.2f}%",
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")


    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.tight_layout()
    plt.show()
```

[26]:
```
# plot confusion matrix for error analysis
y_true_val, y_pred_val = get_ture_and_pred_labels(Validation, CNN1)

print(classification_report(y_true_val, y_pred_val), '\n\n')
cm = confusion_matrix(y_true_val, y_pred_val)

classes = {
    "Arborio":0,
    "Basmati":1,
    "Ipsala":2,
    "Jasmine":3,
    "Karacadag":4,
}

plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues)
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.97      0.95      3000
         1.0       0.96      0.99      0.97      3000
         2.0       1.00      0.98      0.99      3000
         3.0       0.98      0.94      0.96      3000
         4.0       0.98      0.96      0.97      3000

    accuracy                           0.97     15000
   macro avg       0.97      0.97      0.97     15000
weighted avg       0.97      0.97      0.97     15000
```




```
# Save file(HDF5 format)
CNN.save('model_architecture.h5')
```


```
# Save file(HDF5 format)
CNN.save('model_architecture.h5')
```

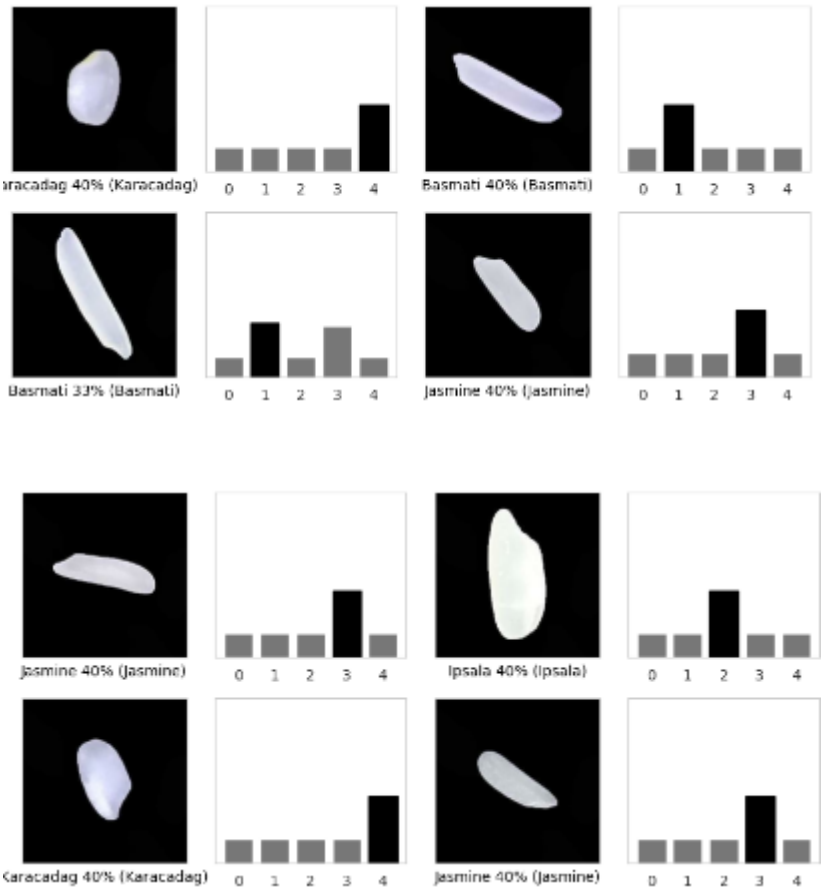

```
# select best model
# Recreate the exact same model, including its weights and the optimizer
Best_model = tf.keras.models.load_model('model_architecture.h5')
```

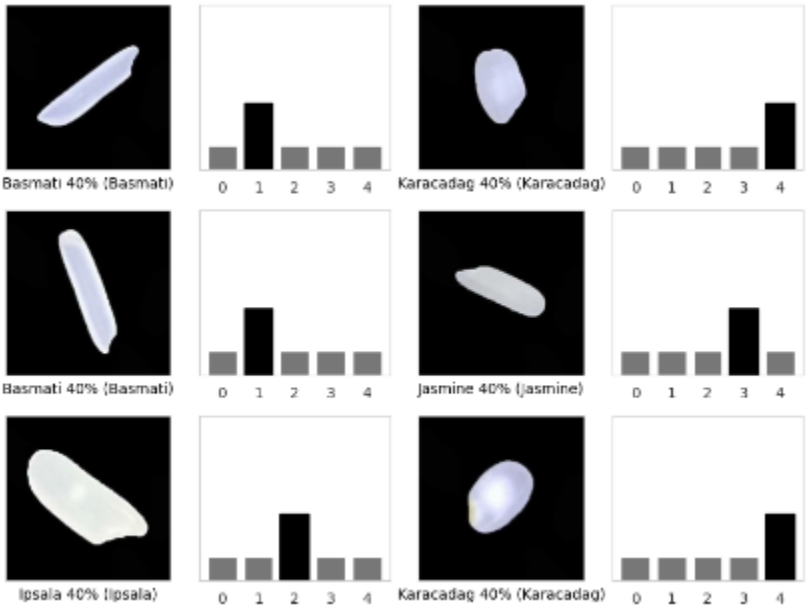
```
def plot_value_array(i, predictions_array, true_label):
    """
    Plot bar plot of predciton
        i (int): image number
        predictions_array (tensor): prediction array of image
        ture_label (tensor): true label for image
    """

    true_label = true_label[i]
    plt.grid(False)
    plt.xticks(range(5))
    plt.yticks([])
    thisplot = plt.bar(range(5), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('black')



#############################################################################################
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img.numpy().astype("uint8"))
    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'black'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)


#############################################################################################
```

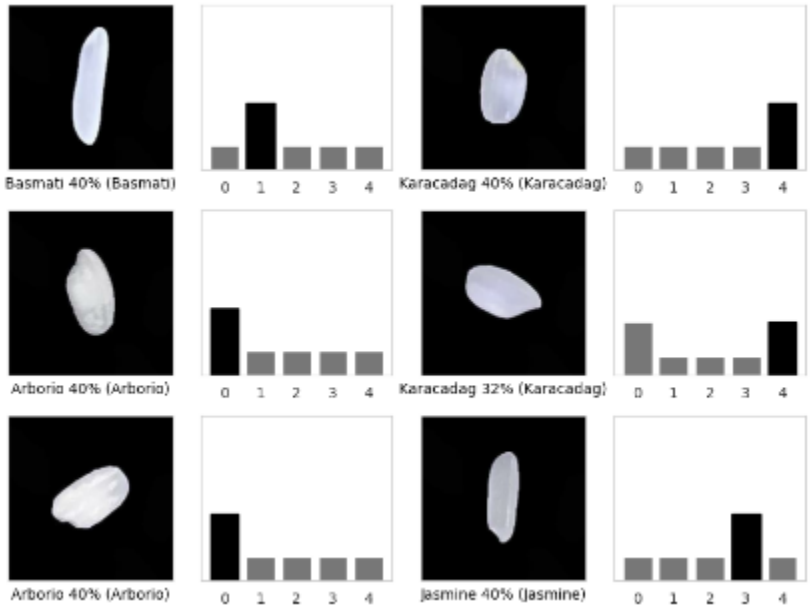
```
def plot_final_result(predictions_array, test_labels, test_images, num_rows=10, num_cols=3):
    """
    Plot the first X test images, their predicted labels, and the true labels.
        predictions_array(): all predictions for all test_image
        test_labels(): all true label for all test_image
        test_images(): all test_image
    """

    num_images = num_rows*num_cols
    plt.figure(figsize=(2*2*num_cols, 2*num_rows))

    for i in range(num_images):
        plt.subplot(num_rows, 2*num_cols, 2*i+1)
        plot_image(i, predictions_array[i], test_labels, test_images)
        plt.subplot(num_rows, 2*num_cols, 2*i+2)
        plot_value_array(i, predictions_array[i], test_labels)

    plt.tight_layout()
    plt.show()
```

```
plot_final_result(predictions, test_labels, test_images, num_rows=5, num_cols=4)
```

Model prediction visualization




Basmati 40% (Basmati)
0 1 2 3 4
Karacadag 40% (Karacadag)
0 1 2 3 4
Basmati 40% (Basmati)
0 1 2 3 4
Jasmine 40% (jasmine)
0 1 2 3 4
Ipsala 40% (Ipsala)
0 1 2 3 4
Karacadag 40% (Karacadag)
0 1 2 3 4


Basmati 40% (Basmati)
0 1 2 3 4
Karacadag 40% (Karacadag)
0 1 2 3 4
Arborio 40% (Arborio)
0 1 2 3 4
Karacadag 32% (Karacadag)
0 1 2 3 4
Arborio 40% (Arborio)
0 1 2 3 4
Jasmine 40% (Jasmine)
0 1 2 3 4