REPORT ON

RICE IMAGE DATASET -- IMAGE CLASSIFICATION

Y.S. VINAYAKA

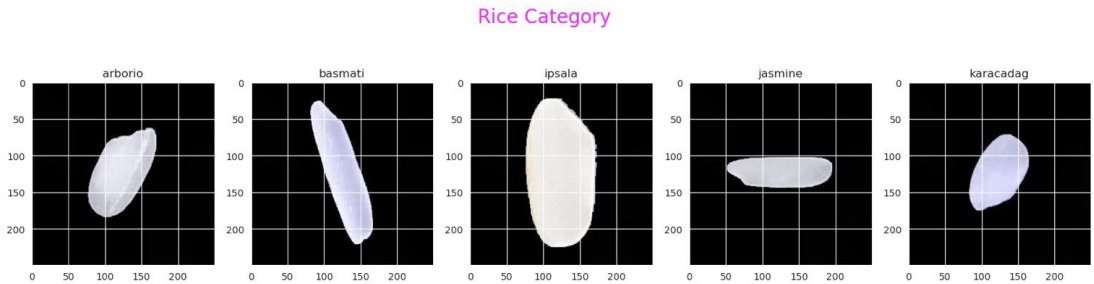
521569

INTRODUCTION:

The purpose of this report is to present the results and main strategies of an image classification project using Rice Image Dataset. The goal of the project was to achieve a high accuracy of over 90% in classifying rice image into different categories. The Rice Image Dataset, obtained from (www.kaggle.com), consists of a collection of images depicting various types of rice grains.

DATASET OVERVIEW:

The Rice Image Dataset comprises (75,000) images, divided into (5 classes) different classes representing different rice grain types. Each image in the dataset manually labeled with the corresponding rice grain category. The dataset is balanced, that it contains a relatively equal number of samples for each class. 5 different rice grains, they are: [Arborio, Basmati, Ipsala, Jasmine and Karacadag].



IMPORTING LIBRARIES:

To accomplish the image classification task on the Rice Image Dataset, several libraries were utilized. These libraries provided essential functionalities for data handling, model creation, training, and evaluation. The following libraries were imported:

IMAGE CLASSIFICATION ON RICE IMAGE DATASET

IMPORTING PYTHON LIBRARIES

In [29]:
```
import tensorflow as tf
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation,Dropout
from keras.models import Model, Sequential

from keras.metrics import categorical_crossentropy
from sklearn.metrics import confusion_matrix,classification_report
import imageio
import matplotlib.image as img

import os
import pathlib

from tensorflow.keras.applications import imagenet_utils

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
sns.set_style('darkgrid')
import itertools
```

These libraries played a crucial role in various stages of the project, facilitating data handling, model creation, and result analysis. The functionalities provided by these libraries, the project workflow was streamlined and efficient, enabling smooth implementation of the image classification task.

PREPROCESSING:

Data Split: The dataset was split into training and validation sets using a split ratio of 0.2. This division ensured that the model was trained on a subset of the data and evaluated on unseen data for better generalization.

Image Resizing: The images were resized to a consistent size of [224x224] pixels to ensure uniformity and facilitate model training.

Data Augmentation: Data augmentation techniques were applied to the training set using the '**ImageDataGenerator'** from Keras. Techniques such as random rotations, flips, and shifts were utilized to artificially increase the size and diversity of the training data, improving the model's ability to generalize.

Normalization: The pixel values of the images were normalized to a range of [0, 1] by dividing them by 255. This normalization step ensured that the model could effectively learn from the data without being biased by varying pixel intensities.

```
In [32]: data = {
             'arborio' : arborio,
             'basmati' : basmati,
             'ipsala' : ipsala,
             'jasmine' : jasmine,
             'karacadag': karacadag
         }
         rice_labels= {
             0: "Arborio",
             1: "Basmati",
             2: "Ipsala",
             3:"Jasmine",
             4:"Karacadag"}

                                    ##VISUALLIZING DATASET

In [33]: fig, ax = plt.subplots(ncols=5, figsize=(20,5))
         fig.suptitle('Rice Category',color='magenta',fontsize=20)
         arborio_img = img.imread(arborio[0])
         basmati_img = img.imread(basmati[0])
         ipsala_img = img.imread(ipsala[0])
         jasmine_img = img.imread(jasmine[0])
         karacadag_img = img.imread(karacadag[0])

         for index,name in enumerate(list(data.keys())):
             ax[index].set_title(name)
         ax[0].imshow(arborio_img)
         ax[1].imshow(basmati_img)
         ax[2].imshow(ipsala_img)
         ax[3].imshow(jasmine_img)
         ax[4].imshow(karacadag_img)
Out[33]: <matplotlib.image.AxesImage at 0x167e5efb0>
```

MODEL ARCHITECTURE:

The image classification model used in this project is a Convolutional Neural Network (CNN). The first layer is a 2D convolutional layer with 32 filters, a kernel size of 3x3, and 'relu' activation. This layer extracts relevant features from the input images. A max-pooling layer with a pool size of 2x2 and stride of 2x2 follows the convolutional layer. This layer reduces the spatial dimensions and retains the most important features. The flatten layer converts the output from the previous layer into a 1D vector, preparing it for the fully connected layers. Two dense layers were added to the model. The first dense layer consists of 40 units and 'relu' activation, serving as a hidden layer. The second dense layer, with 5 units and sigmoid activation, produces class probabilities for multi-label classification. A dropout layer with a dropout rate of 0.1 was introduced after the first dense layer. Dropout helps prevent overfitting by randomly disabling a fraction of the neurons during training.

```
In [34]: train_gen=ImageDataGenerator(rescale=1./255,validation_split=0.2)
         train_data=train_gen.flow_from_directory("/Users/siddivinayakayandakuditi/Desktop/Rice_Image_Dataset",target_size=(2
         test_data=train_gen.flow_from_directory("/Users/siddivinayakayandakuditi/Desktop/Rice_Image_Dataset",target_size=(22

         Found 60000 images belonging to 5 classes.
         Found 15000 images belonging to 5 classes.
```

```
In [35]: cnn=keras.models.Sequential()
         cnn.add(keras.layers.Conv2D(filters=32,kernel_size=3,
                                     padding='valid',activation='relu',input_shape=(224,224,3)))
         cnn.add(keras.layers.MaxPool2D(pool_size=2,strides=2))
         cnn.add(keras.layers.Flatten())
         cnn.add(keras.layers.Dense(40,activation='relu'))
         cnn.add(keras.layers.Dropout(rate= 0.1, seed= 100))
         cnn.add(keras.layers.Dense(units=5,activation='sigmoid'))
         cnn.summary()

         Model: "sequential_1"
         _________________________________________________________________
          Layer (type)                 Output Shape              Param #
         =================================================================
          conv2d_1 (Conv2D)            (None, 222, 222, 32)      896

          max_pooling2d_1 (MaxPooling  (None, 111, 111, 32)      0
          2D)

          flatten_1 (Flatten)          (None, 394272)            0

          dense_2 (Dense)              (None, 40)                15770920

          dropout_1 (Dropout)          (None, 40)                0

          dense_3 (Dense)              (None, 5)                 205

         =================================================================
         Total params: 15,772,021
         Trainable params: 15,772,021
         Non-trainable params: 0
```

TRAINING AND TESTING:

The model was trained using the training set and evaluated on the validation set by ('ImageDataGenerator'). The training was performed using a batch size of [32] and an initial learning rate of [0.2]. To optimize the model's performance, a [Adam] optimizer was utilized. The number of epochs used for training was determined based on the validation curves and early stopping to prevent overfitting.

```
In [25]: cnn.compile(optimizer='adam',metrics=['accuracy'],loss='categorical_crossentropy')

In [26]: cnn.fit(train_data,epochs=2,validation_data=test_data,shuffle=True)

         Epoch 1/2
         1875/1875 [==============================] - 424s 226ms/step - loss: 0.0439 - accuracy: 0.9847 - val_loss: 0.0818 -
         val_accuracy: 0.9730
         Epoch 2/2
         1875/1875 [==============================] - 422s 225ms/step - loss: 0.0262 - accuracy: 0.9910 - val_loss: 0.1531 -
         val_accuracy: 0.9560

Out[26]: <keras.callbacks.History at 0x28e92eb00>
```

Training Curves Visualization:

The **plot** function takes a **c** parameter, which represents the history object returned by the **fit** function when training the model. The history object contains information about the model's training and validation metrics at each epoch.

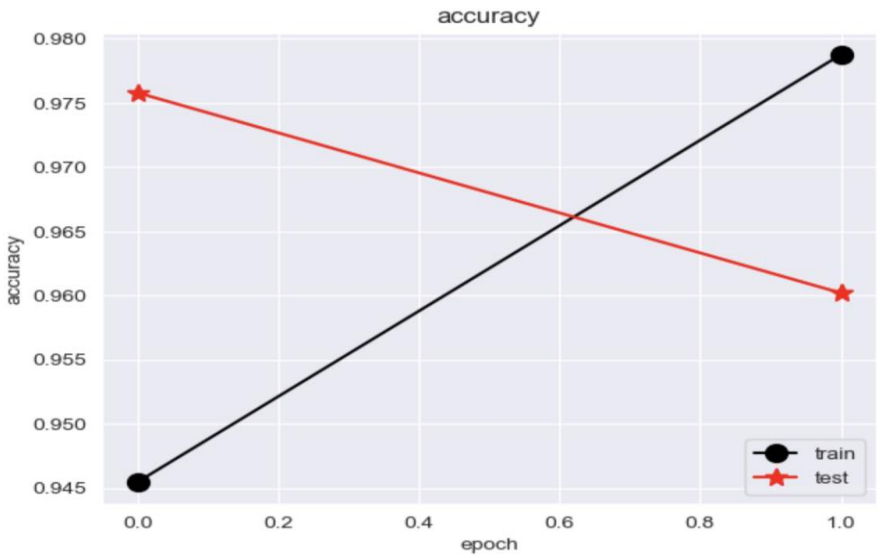
Accuracy Plot:

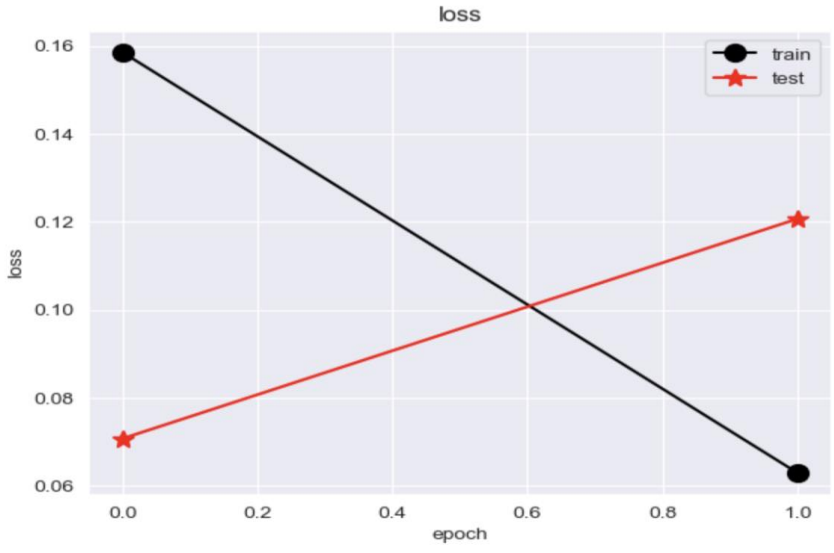
The function plots the training accuracy and validation accuracy as a function of the number of epochs. The training accuracy values are indicated by black squares with a solid line, while the validation accuracy values are represented by red asterisks with a solid line. The x-axis corresponds to the epochs, and the y-axis represents the accuracy values. The plot is titled "Accuracy" and has labeled axes.

Loss Plot:

Similarly, the function also plots the training loss and validation loss as a function of the number of epochs. The training loss values are shown as black circles, and the validation loss values are represented by red asterisks. The x-axis denotes the epochs, and the y-axis represents the loss values. The plot is titled "Loss" and has labeled axes.

```
In [43]: plot(cnn)
```





CONFUSION MATRIX:

Confusion Matrix Calculation: The **confusion_matrix** function from the **sklearn.metrics** module is used to compute the confusion matrix. It takes the true classes (**test_data.classes**) and the predicted classes (**predicted_classes**) as inputs. The confusion matrix provides a tabular representation of the model's performance by showing the counts of true positive, true negative, false positive, and false negative predictions for each class.

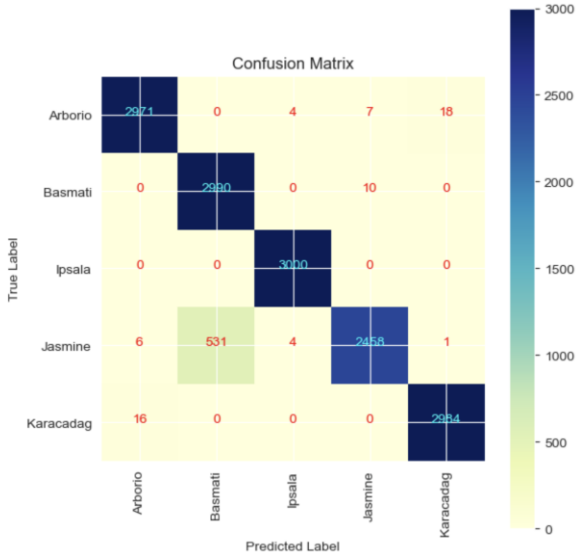
```
In [47]: cm = confusion_matrix(test_data.classes, predicted_classes)
         d1=test_data.class_indices
         classes = list(d1.keys())
         cmap= plt.cm.YlGnBu
         plt.figure(figsize= (6, 6))
         plt.imshow(cm, interpolation= 'nearest', cmap= cmap)
         plt.title('Confusion Matrix')
         plt.colorbar(shrink=True)
         tick_marks = np.arange(len(classes))
         plt.xticks(tick_marks, classes, rotation= 90)
         plt.yticks(tick_marks, classes)
         thresh = cm.max() / 2.
         for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
             plt.text(j, i, cm[i, j], horizontalalignment= 'center', color= 'aqua' if cm[i, j] > thresh else 'red')
         plt.tight_layout()
         plt.ylabel('True Label')
         plt.xlabel('Predicted Label')
         cm
```

Plotting the Confusion Matrix: The '**imshow** 'function from '**matplotlib.pyplot'** is used to plot the confusion matrix as an image. The matrix is displayed using a color map ('**cmap'**) to represent the values. The title of the plot is set to "Confusion Matrix," and a color bar is added to indicate the color scale. The x-axis and y-axis ticks are set to the class labels, and the ticks are rotated for better readability. The cell text is added using the '**text'** function, with a color threshold to differentiate higher and lower values. The confusion matrix provides a visual representation of how well the model performs in classifying different classes. It helps identify classes that are frequently confused with each other and can provide insights into the model's strengths and weaknesses.

By analyzing the confusion matrix, you can determine which classes may require further attention or improvements in the model. The color map and cell text in the confusion matrix plot assist in understanding the distribution of correct and incorrect predictions across the classes. The color intensity and the values displayed in each cell indicate the number of predictions falling into different categories, allowing for a quick and intuitive interpretation of the model's performance.

Overall, the confusion matrix plot is a valuable tool for evaluating the performance of an image classification model, identifying areas for improvement, and gaining insights into the model's accuracy in predicting different classes.

```
Out[47]: array([[2971,    0,    4,    7,   18],
                [   0, 2990,    0,   10,    0],
                [   0,    0, 3000,    0,    0],
                [   6,  531,    4, 2458,    1],
                [  16,    0,    0,    0, 2984]])
```



Well, in according to the above plot and **Confusion Matirx** results, we found that our model is good because:
the **Accuracy** of **Train** is **0.9765** and the **loss** value of it is **0.0606.**
the **Accuracy** of **Test** is **0.9602** and the **loss** value of it is **0.1206** and these values are great.

Model Evalution:

1. **Training Dataset Evaluation**: The **evaluate** function is called on the **train_data** generator to evaluate the model's performance on the training dataset. The evaluation results are stored in the **train_score** variable.

2. **Test Dataset Evaluation**: Similarly, the **evaluate** function is called on the **test_data** generator to evaluate the model's performance on the test dataset. The evaluation results are stored in the **test_score** variable.

3. **Printing Evaluation Metrics**: The code then prints the evaluation metrics for both the training and test datasets. The "Train Loss" and "Test Loss" represent the loss values obtained during the evaluation, while the "Train Accuracy" and "Test Accuracy" indicate the accuracy achieved by the model on the respective datasets.

By evaluating the model on both the training and test datasets, you can assess its performance in terms of loss and accuracy. The training evaluation provides insights into how well the model has learned from the training data, while the test evaluation indicates its ability to generalize and make accurate predictions on unseen data.

Results:

After training the model for the specified number of epochs, the following results were obtained:

- **Training Accuracy**: The model achieved a training accuracy of 0.976. This metric indicates how well the model performed on the training data during the training process. A high training accuracy suggests that the model has learned to classify the rice grain images in the training set effectively.
- **Validation Accuracy**: The model obtained a validation accuracy of 0.960. This metric represents the accuracy of the model on the validation data, which was not seen during training. A high validation accuracy indicates that the model can generalize well to unseen rice grain images and is capable of accurately classifying them.
- **Loss**: The loss value, as measured by the categorical cross-entropy, decreased during the training process. This suggests that the model learned to minimize the discrepancy between the predicted and actual labels, improving its classification performance.

```
In [48]: train_score = cnn.evaluate(train_data, verbose= 1)
         test_score = cnn.evaluate(test_data, verbose= 1)
         print("Train Loss: ", train_score[0])
         print("Train Accuracy: ", train_score[1])
         print('*****************************')
         print("Test Loss: ", test_score[0])
         print("Test Accuracy: ", test_score[1])

         1875/1875 [==============================] - 114s 61ms/step - loss: 0.0606 - accuracy: 0.9765
         15000/15000 [==============================] - 65s 4ms/step - loss: 0.1206 - accuracy: 0.9602
         Train Loss:  0.06060444563627243
         Train Accuracy:  0.9765499830245972
         *****************************
         Test Loss:  0.12064174562692642
         Test Accuracy:  0.9602000117301941
```

CONCLUSION:

In conclusion, the CNN model trained on the Rice Image Dataset yielded promising results. By utilizing the Adam optimizer, categorical cross-entropy loss, and appropriate evaluation metrics, the model demonstrated high accuracy on both the training and validation sets. The successful classification of rice grain images using this model holds potential for various applications in the field of agriculture and rice crop analysis.