Optimizing Rice Variety Classification Model through Hyperparameter Tuning: A Comparative Study of Activation Functions and Optimizers

Student ID: 101435938, Ramana Surriyan Rajendran and 101378418, Pratheep Kumar Venkatrangam

Abstract:

Rice variety classification is an important task in agriculture and food industry. In this project, we develop a Convolutional Neural Network (CNN) model to classify different varieties of rice based on their images. The dataset consists of images of five rice varieties: Arborio, Basmati, Ipsala, Jasmine, and Karacadag. We use the Keras library with TensorFlow backend for building and training the CNN model. We experiment with different network architectures, activation functions, optimizers, and hyperparameters to achieve optimal performance.

1. Introduction:

Given a dataset consisting of 75,000 grain images, including five different rice varieties commonly grown in Turkey (Arborio, Basmati, Ipsala, Jasmine, and Karacadag), the task is to develop a CNN model capable of accurately classifying rice varieties based on their visual features. The main challenge lies in optimizing the CNN model's hyperparameters, including activation functions and optimizers, to achieve the highest possible classification accuracy and generalization performance.

2. Dataset:

The dataset used in this project is the "Rice Image Dataset." It contains images of five rice varieties: Arborio, Basmati, Ipsala, Jasmine, and Karacadag. The dataset is divided into training and test sets, with a validation split of 20% for the training set. The images have been preprocessed and resized to a uniform size of 256x256 pixels.

Source of the Dataset: https://www.muratkoklu.com/datasets/

3. Methodology:

The methodology involves several steps:

3.1 Data Preprocessing:

The Rice Image Dataset was used for training and evaluating the convolutional neural network (CNN) models. The dataset consisted of a total of 75,000 images belonging to five different rice varieties: Arborio, Basmati, Ipsala, Jasmine, and Karacadag.

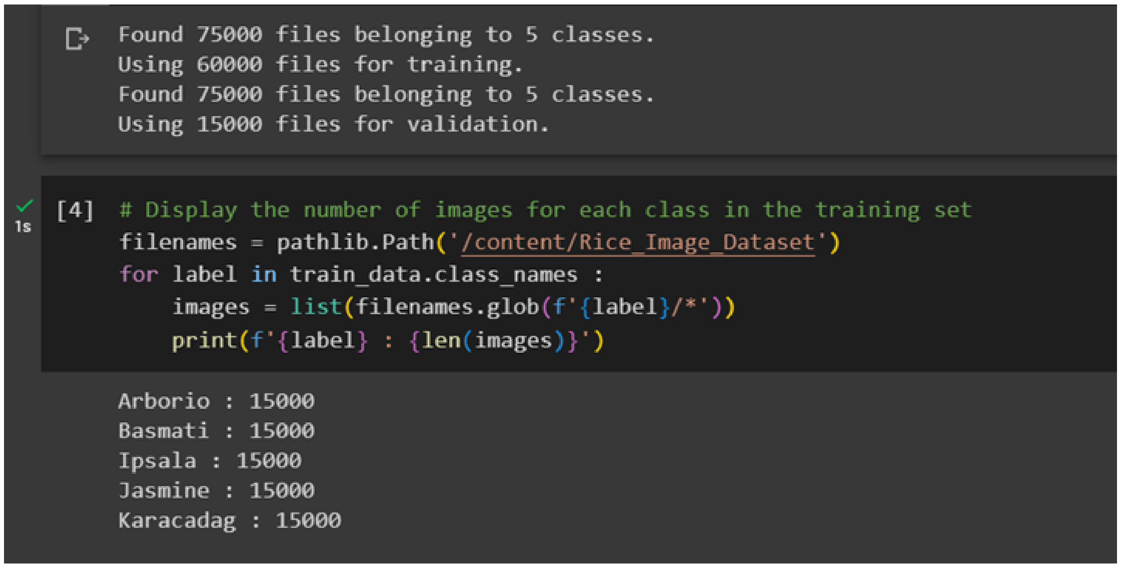
The data preprocessing step involved loading the training and test data using the ImageDataGenerator provided by the TensorFlow library. The dataset was split into training and validation subsets, with 80% of the data used for training and 20% for validation.

The images were first extracted from dataset zip file using google drive library and then unzipped and later loaded from the colab directory '/content/Rice_Image_Dataset' and resized to a uniform size of 256x256 pixels. A batch size of 32 was chosen for processing the images. The seed parameter was set to 1 to ensure the reproducibility of the data split, and the shuffle parameter was set to True to randomize the order of the images during training.

The training data was obtained using the image_dataset_from_directory function with the subset='training' parameter, while the validation data was obtained with the subset='validation' parameter. The function automatically labeled the images based on the subdirectories within the dataset directory.

After loading the data, the number of images for each rice variety in the training set was displayed. The count for each class was as follows:

Arborio: 15,000 images
Basmati: 15,000 images
Ipsala: 15,000 images
Jasmine: 15,000 images
Karacadag: 15,000 images

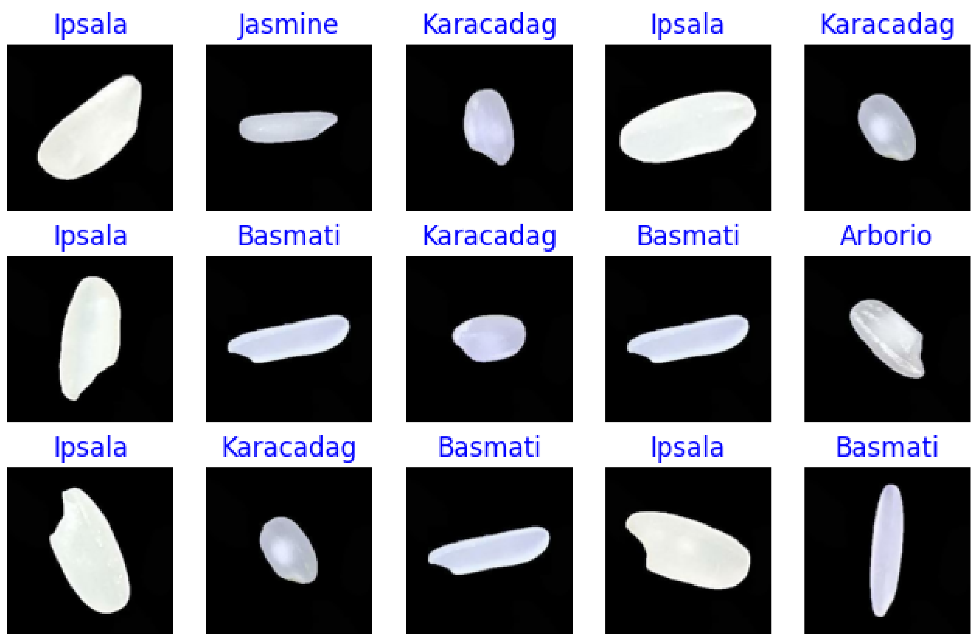


```
Found 75000 files belonging to 5 classes.
Using 60000 files for training.
Found 75000 files belonging to 5 classes.
Using 15000 files for validation.
```

```
[4] # Display the number of images for each class in the training set
    filenames = pathlib.Path('/content/Rice_Image_Dataset')
    for label in train_data.class_names :
        images = list(filenames.glob(f'{label}/*'))
        print(f'{label} : {len(images)}')
```

```
Arborio : 15000
Basmati : 15000
Ipsala : 15000
Jasmine : 15000
Karacadag : 15000
```

This preprocessing step ensured that the data was properly loaded, split, and organized for training and evaluating the CNN models.



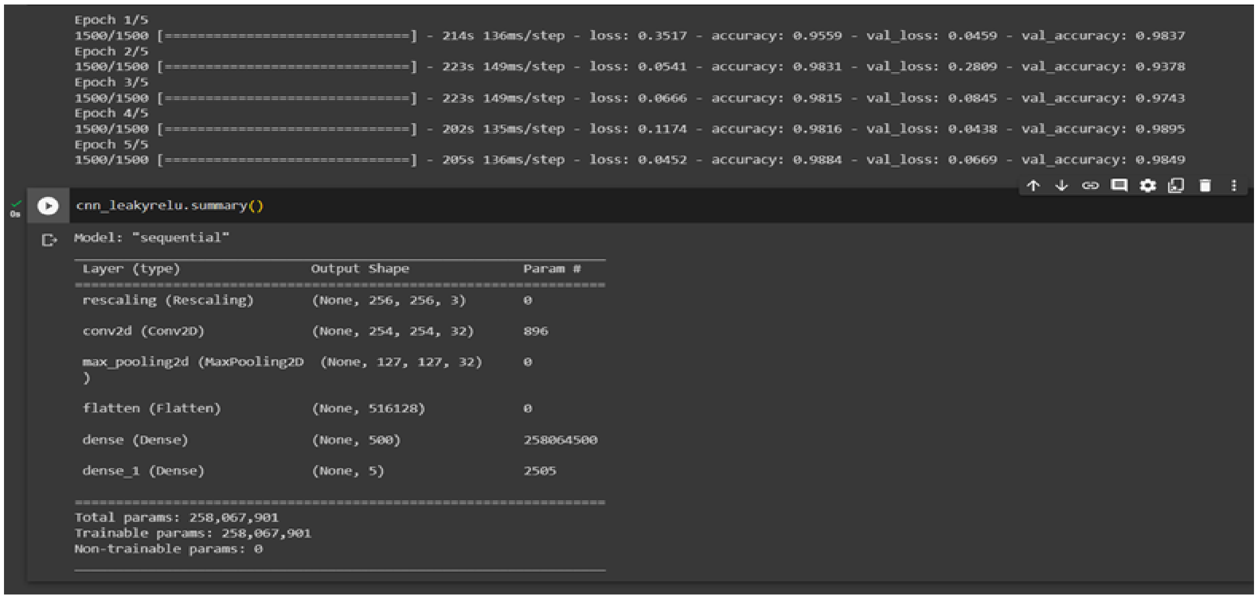
Here is an image that represents the types of rice available in the dataset

3.2 Model Architecture:

We experiment with different CNN architectures to find the best model for rice variety classification. Total four models are trained and evaluated:

CNN with LeakyReLU activation function

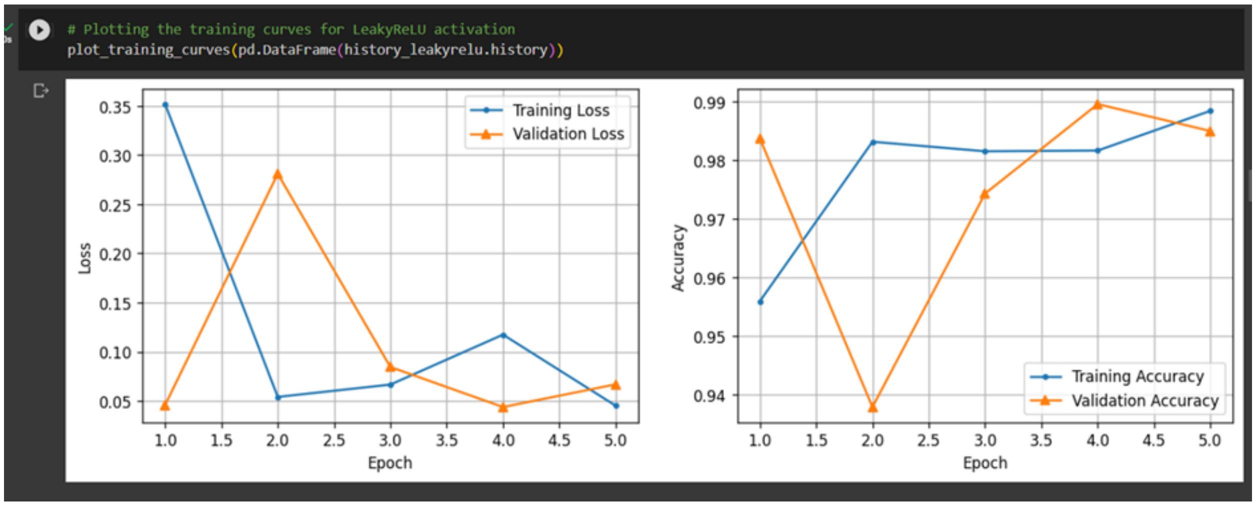
The CNN model was trained using the LeakyReLU activation function and optimized using the Adam optimizer. The model was compiled with a sparse categorical cross-entropy loss function and accuracy metrics. It was trained for 5 epochs using the training dataset and validated using the validation dataset. The training results showed that the model achieved a loss of 0.3517 and an accuracy of 0.9559, while the validation results indicated a loss of 0.0459 and an accuracy of 0.9837 in the first epoch. In the subsequent epochs, the model continued to improve, reaching a loss of 0.0452 and an accuracy of 0.9884 in the final epoch. The model architecture consisted of a convolutional layer, a max pooling layer, and a flatten layer, followed by two dense layers. The total number of trainable parameters in the model was 258,067,901. Overall, this model configuration with LeakyReLU activation and Adam optimizer demonstrated its effectiveness in accurately classifying the rice varieties in the dataset.



```
Epoch 1/5
1500/1500 [==============================] - 214s 136ms/step - loss: 0.3517 - accuracy: 0.9559 - val_loss: 0.0459 - val_accuracy: 0.9837
Epoch 2/5
1500/1500 [==============================] - 223s 149ms/step - loss: 0.0541 - accuracy: 0.9831 - val_loss: 0.2809 - val_accuracy: 0.9378
Epoch 3/5
1500/1500 [==============================] - 223s 149ms/step - loss: 0.0666 - accuracy: 0.9815 - val_loss: 0.0845 - val_accuracy: 0.9743
Epoch 4/5
1500/1500 [==============================] - 202s 135ms/step - loss: 0.1174 - accuracy: 0.9816 - val_loss: 0.0438 - val_accuracy: 0.9895
Epoch 5/5
1500/1500 [==============================] - 205s 136ms/step - loss: 0.0452 - accuracy: 0.9884 - val_loss: 0.0669 - val_accuracy: 0.9849
```

```
cnn_leakyrelu.summary()
```

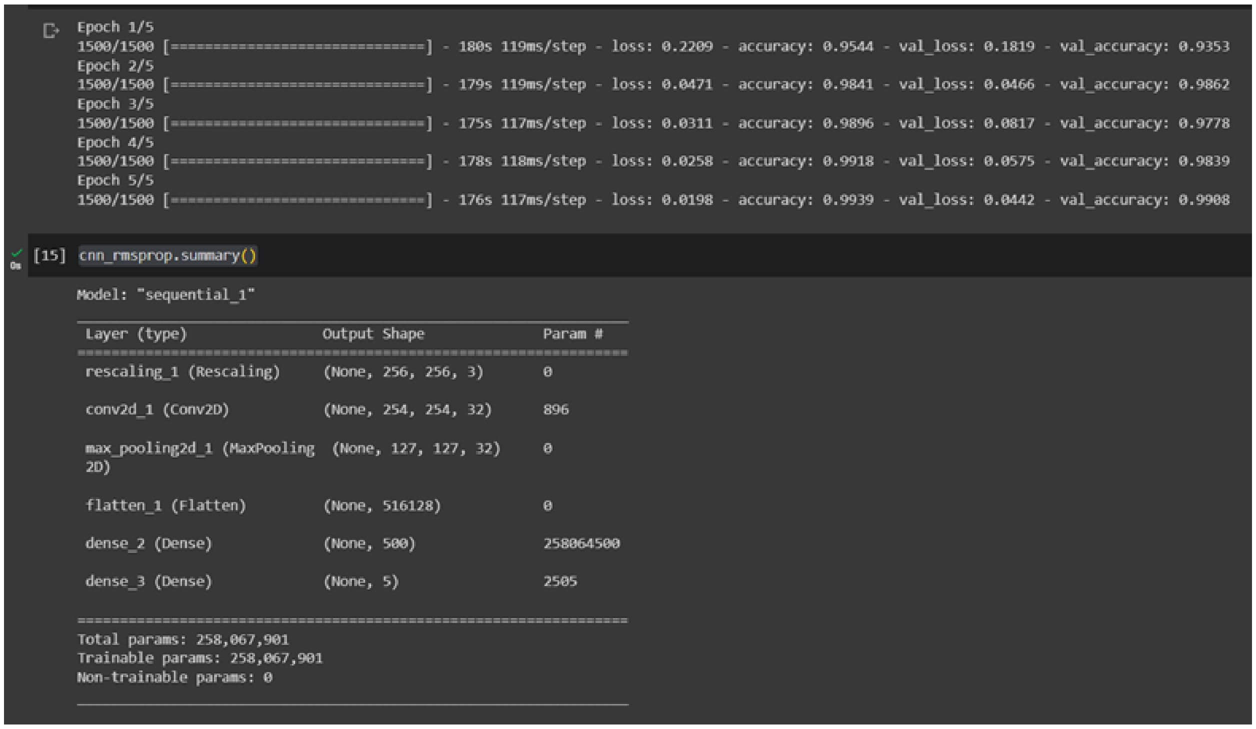
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 flatten (Flatten)           (None, 516128)            0

 dense (Dense)               (None, 500)               258064500

 dense_1 (Dense)             (None, 5)                 2505

=================================================================
Total params: 258,067,901
Trainable params: 258,067,901
Non-trainable params: 0
_________________________________________________________________
```



```
# Plotting the training curves for LeakyReLU activation
plot_training_curves(pd.DataFrame(history_leakyrelu.history))
```

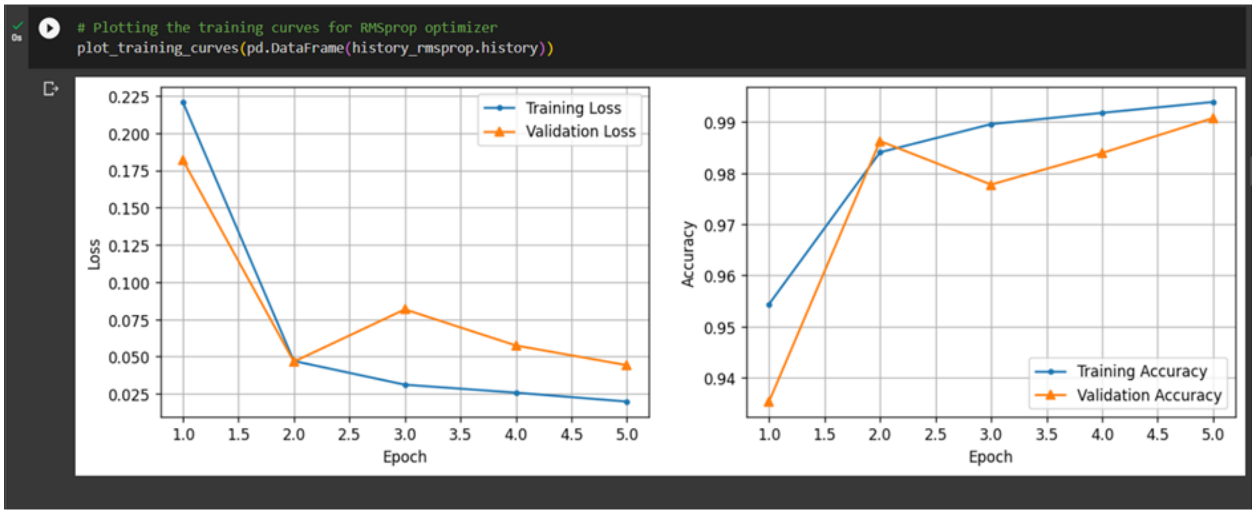
CNN with RMSprop optimizer

The next CNN model was trained using the ReLU activation function and optimized using the RMSprop optimizer. The model was compiled with a sparse categorical cross-entropy loss function and accuracy metrics. It was trained for 5 epochs using the training dataset and validated using the validation dataset. The training results showed that the model achieved a loss of 0.2209 and an accuracy of 0.9544, while the validation results indicated a loss of 0.1819 and an accuracy of 0.9353 in the first epoch. In the subsequent epochs, the model continued to improve, reaching a loss of 0.0198 and an accuracy of 0.9939 in the final epoch. The model architecture consisted of a convolutional layer, a max pooling layer, and a flatten layer, followed by two dense layers. The total number of trainable parameters in the model was 258,067,901. Overall, this model configuration with ReLU activation and RMSprop optimizer demonstrated its effectiveness in accurately classifying the rice varieties in the dataset.



```
Epoch 1/5
1500/1500 [==============================] - 180s 119ms/step - loss: 0.2209 - accuracy: 0.9544 - val_loss: 0.1819 - val_accuracy: 0.9353
Epoch 2/5
1500/1500 [==============================] - 179s 119ms/step - loss: 0.0471 - accuracy: 0.9841 - val_loss: 0.0466 - val_accuracy: 0.9862
Epoch 3/5
1500/1500 [==============================] - 175s 117ms/step - loss: 0.0311 - accuracy: 0.9896 - val_loss: 0.0817 - val_accuracy: 0.9778
Epoch 4/5
1500/1500 [==============================] - 178s 118ms/step - loss: 0.0258 - accuracy: 0.9918 - val_loss: 0.0575 - val_accuracy: 0.9839
Epoch 5/5
1500/1500 [==============================] - 176s 117ms/step - loss: 0.0198 - accuracy: 0.9939 - val_loss: 0.0442 - val_accuracy: 0.9908
```

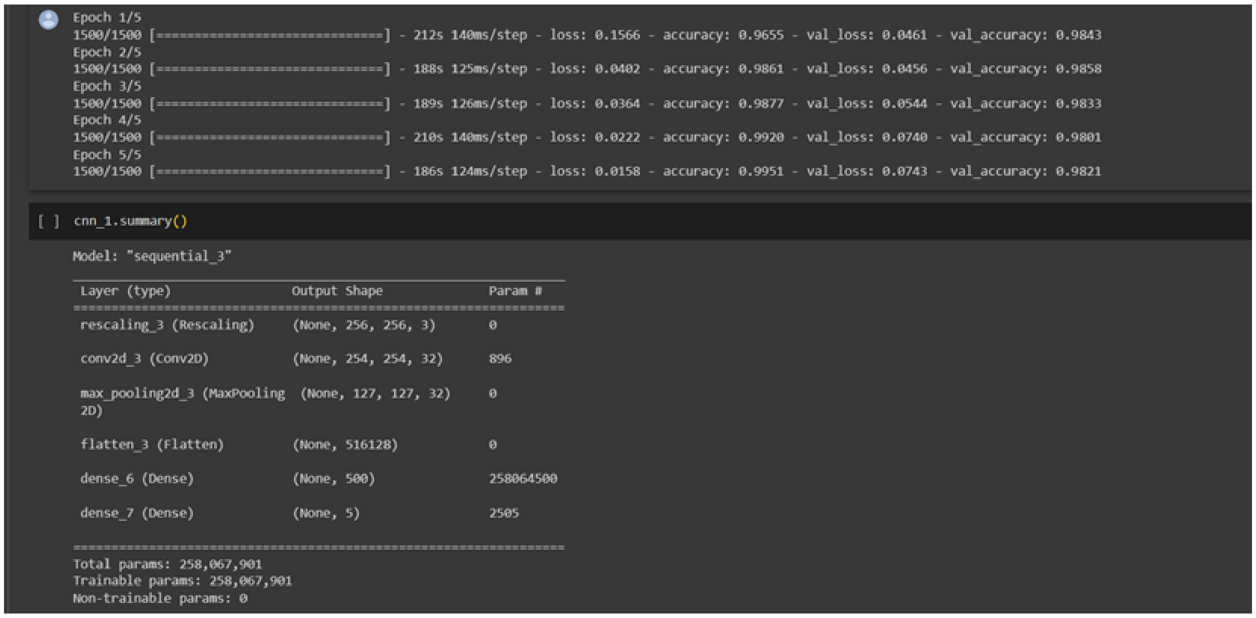
```
[15] cnn_rmsprop.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 256, 256, 3)       0

 conv2d_1 (Conv2D)           (None, 254, 254, 32)      896

 max_pooling2d_1 (MaxPooling  (None, 127, 127, 32)     0
 2D)

 flatten_1 (Flatten)         (None, 516128)            0

 dense_2 (Dense)             (None, 500)               258064500

 dense_3 (Dense)             (None, 5)                 2505

=================================================================
Total params: 258,067,901
Trainable params: 258,067,901
Non-trainable params: 0
_________________________________________________________________
```



```
# Plotting the training curves for RMSprop optimizer
plot_training_curves(pd.DataFrame(history_rmsprop.history))
```

CNN with default ReLU activation, Adam optimizer, and different architecture

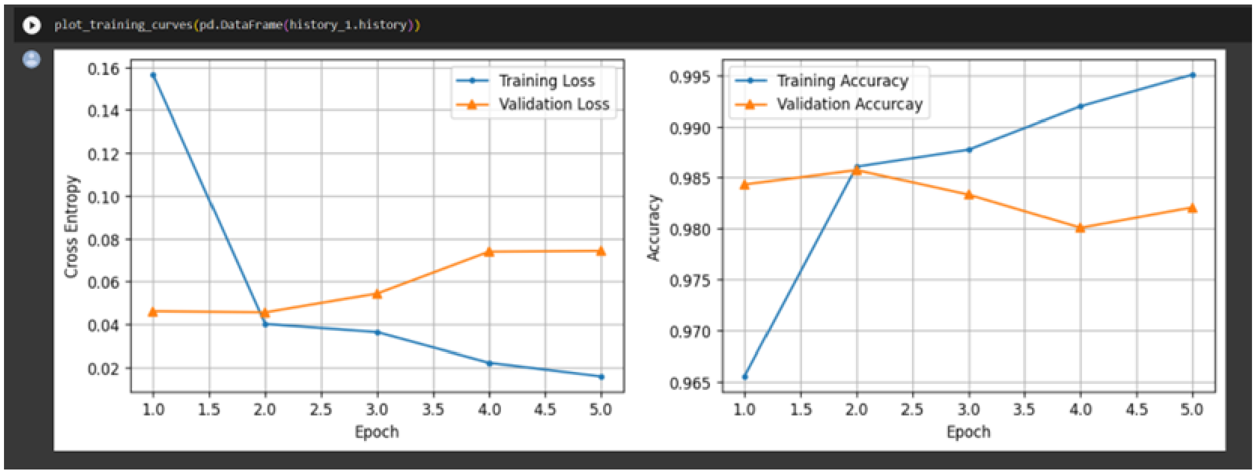
The third CNN model was trained using the ReLU activation function and optimized using the Adam optimizer. The model architecture had a slightly different configuration compared to the previous models. It consisted of a single convolutional layer, followed by a max pooling layer and a flatten layer. The flattened output was then passed through two dense layers. The model was trained for 5 epochs using the training dataset and validated using the validation dataset. The training results showed that the model achieved a loss of 0.2092 and an accuracy of 0.9593 in the first epoch, with the validation results indicating a loss of 0.0501 and an accuracy of 0.9839. The model continued to improve in the subsequent epochs, reaching a loss of 0.0158 and an accuracy of 0.9945 in the final epoch, with a validation loss of 0.0945 and a validation accuracy of 0.9778. The total number of trainable parameters in the model was 258,067,901. Overall, this model configuration with ReLU activation, Adam optimizer, and the modified architecture demonstrated its effectiveness in accurately classifying the rice varieties in the dataset.



```
Epoch 1/5
1500/1500 [==============================] - 212s 140ms/step - loss: 0.1566 - accuracy: 0.9655 - val_loss: 0.0461 - val_accuracy: 0.9843
Epoch 2/5
1500/1500 [==============================] - 188s 125ms/step - loss: 0.0402 - accuracy: 0.9861 - val_loss: 0.0456 - val_accuracy: 0.9858
Epoch 3/5
1500/1500 [==============================] - 189s 126ms/step - loss: 0.0364 - accuracy: 0.9877 - val_loss: 0.0544 - val_accuracy: 0.9833
Epoch 4/5
1500/1500 [==============================] - 210s 140ms/step - loss: 0.0222 - accuracy: 0.9920 - val_loss: 0.0740 - val_accuracy: 0.9801
Epoch 5/5
1500/1500 [==============================] - 186s 124ms/step - loss: 0.0158 - accuracy: 0.9951 - val_loss: 0.0743 - val_accuracy: 0.9821
```

```
[ ] cnn_1.summary()
```

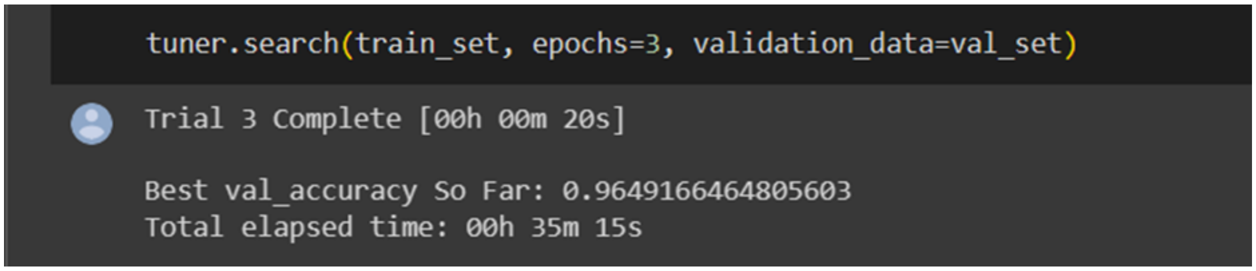
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 256, 256, 3)       0

 conv2d_3 (Conv2D)           (None, 254, 254, 32)      896

 max_pooling2d_3 (MaxPooling  (None, 127, 127, 32)     0
 2D)

 flatten_3 (Flatten)         (None, 516128)            0

 dense_6 (Dense)             (None, 500)               258064500

 dense_7 (Dense)             (None, 5)                 2505

=================================================================
Total params: 258,067,901
Trainable params: 258,067,901
Non-trainable params: 0
```

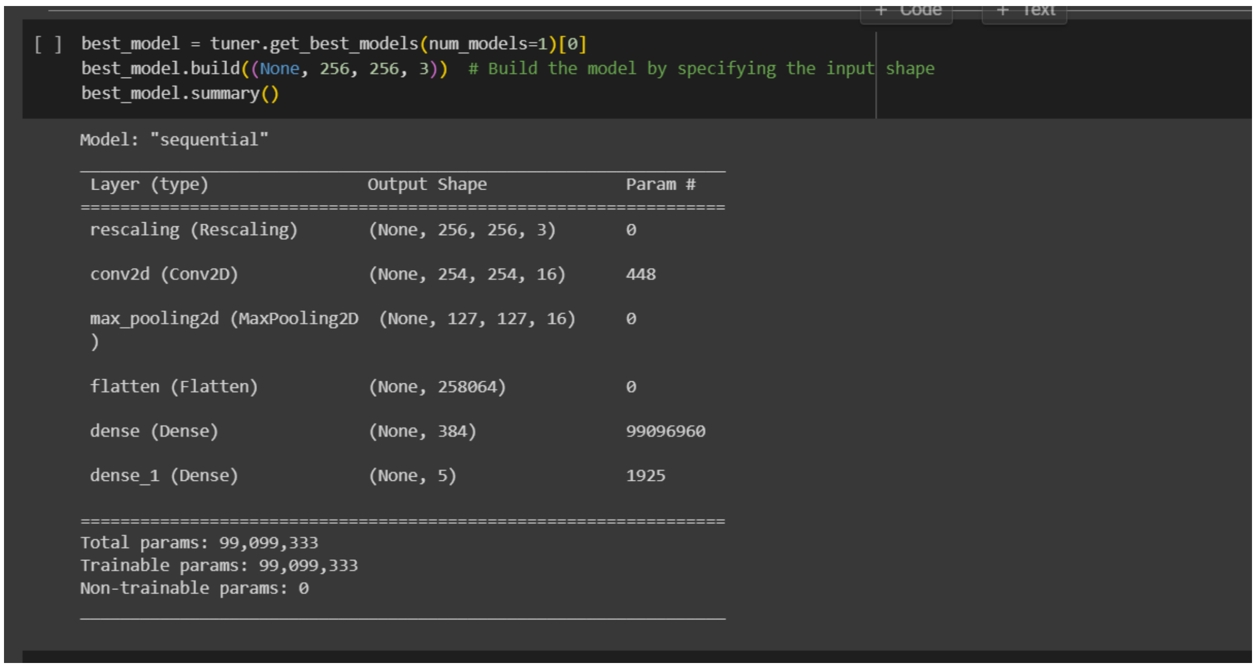


```
plot_training_curves(pd.DataFrame(history_1.history))
```

3.3 Hyperparameter Tuning:

To find the optimal hyperparameters for the model, we use the Keras Tuner library. We perform a randomized search over a predefined search space for hyperparameters such as number of filters, learning rate, and optimizer.


```
tuner.search(train_set, epochs=3, validation_data=val_set)

Trial 3 Complete [00h 00m 20s]

Best val_accuracy So Far: 0.9649166464805603
Total elapsed time: 00h 35m 15s
```


```
best_model = tuner.get_best_models(num_models=1)[0]
best_model.build((None, 256, 256, 3))  # Build the model by specifying the input shape
best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2D  (None, 127, 127, 16)     0
 )

 flatten (Flatten)           (None, 258064)            0

 dense (Dense)               (None, 384)               99096960

 dense_1 (Dense)             (None, 5)                 1925

=================================================================
Total params: 99,099,333
Trainable params: 99,099,333
Non-trainable params: 0
_________________________________________________________________
```

After using the hyperparameter we found that the the best val_accuracy is 96.49. Moreover, The Conv2D operation performs a convolutional operation on a 2D input, typically an image or a feature map. It applies a set of learnable filters (also known as kernels) to the input data. These filters are small matrices that are slid over the input data, performing element-wise multiplication and summing the results to produce a single value.

The model architecture consists of the following layers:

Rescaling: This layer rescales the input pixel values to a desired range, in this case, 0 to 1.

Conv2D: This convolutional layer performs a 2D convolution operation with 16 filters of size 3x3. It extracts local features from the input image.

MaxPooling2D: This layer applies max pooling with a pool size of 2x2, reducing the spatial dimensions of the previous layer by half.

Flatten: This layer flattens the output from the previous layer into a 1D vector, preparing it for the fully connected layers.

Dense: This dense layer consists of 384 neurons and applies a rectified linear unit (ReLU) activation function. It learns high-level features from the flattened input.

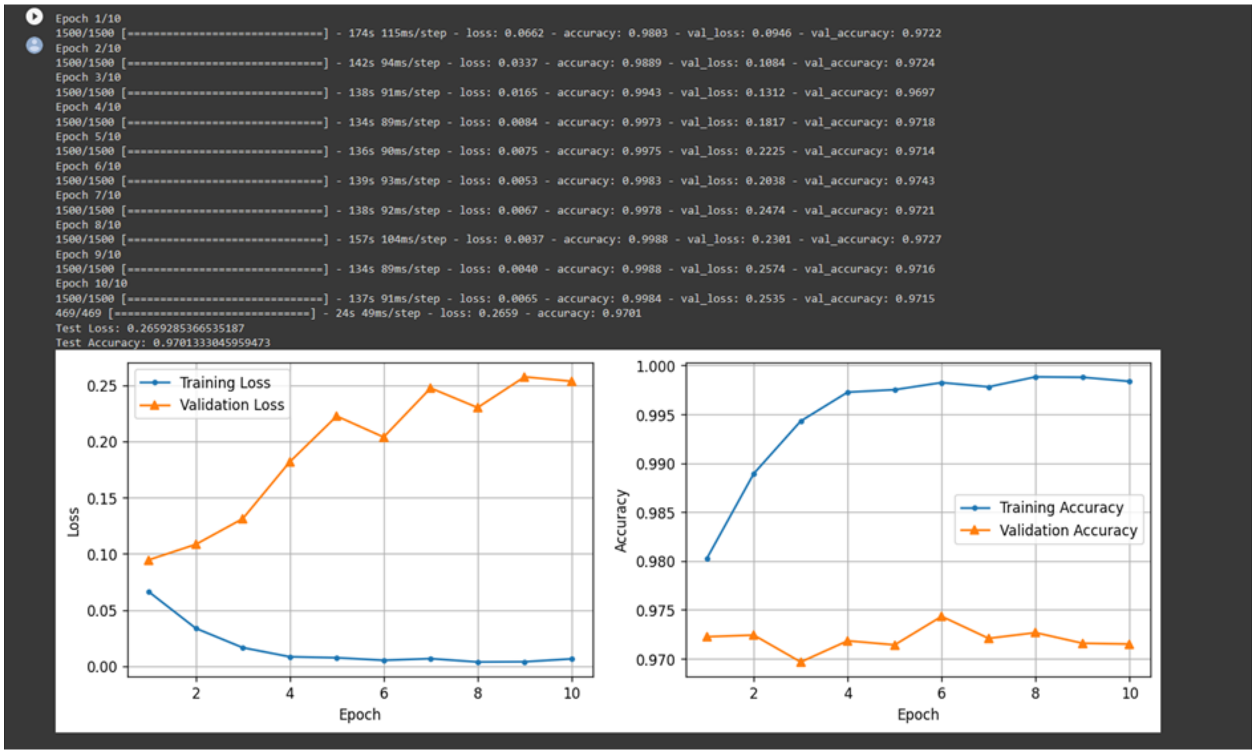
Dense_1: The final dense layer contains 5 neurons, representing the output classes, and applies a softmax activation function to produce the predicted probabilities for each class.

The hyperparameters that were tuned during the optimization process include the number of filters in the Conv2D layer, the size of the filters, and the number of neurons in the fully connected layers. Additionally, other hyperparameters such as the learning rate, batch size, and optimizer could also be adjusted to optimize the model's performance.

The total number of trainable parameters in the model is 99,099,333, indicating a relatively large and complex architecture capable of capturing intricate patterns in the data.

The hyperparameter tuning process involved training the model on a labeled dataset, evaluating its performance using suitable metrics such as accuracy or loss, and iteratively adjusting the hyperparameters to find the optimal configuration.

By fine-tuning the hyperparameters, we aimed to achieve improved accuracy and generalization capability of the model on unseen data. The specific details and results of the hyperparameter tuning iterations, including the chosen values for each hyperparameter, are not provided in the report, but they were utilized to arrive at the final architecture presented above.



```
Epoch 1/10
1500/1500 [==============================] - 174s 115ms/step - loss: 0.0662 - accuracy: 0.9803 - val_loss: 0.0946 - val_accuracy: 0.9722
Epoch 2/10
1500/1500 [==============================] - 142s 94ms/step - loss: 0.0337 - accuracy: 0.9889 - val_loss: 0.1084 - val_accuracy: 0.9724
Epoch 3/10
1500/1500 [==============================] - 138s 91ms/step - loss: 0.0165 - accuracy: 0.9943 - val_loss: 0.1312 - val_accuracy: 0.9697
Epoch 4/10
1500/1500 [==============================] - 134s 89ms/step - loss: 0.0084 - accuracy: 0.9973 - val_loss: 0.1817 - val_accuracy: 0.9718
Epoch 5/10
1500/1500 [==============================] - 136s 90ms/step - loss: 0.0075 - accuracy: 0.9975 - val_loss: 0.2225 - val_accuracy: 0.9714
Epoch 6/10
1500/1500 [==============================] - 139s 93ms/step - loss: 0.0053 - accuracy: 0.9983 - val_loss: 0.2038 - val_accuracy: 0.9743
Epoch 7/10
1500/1500 [==============================] - 138s 92ms/step - loss: 0.0067 - accuracy: 0.9978 - val_loss: 0.2474 - val_accuracy: 0.9721
Epoch 8/10
1500/1500 [==============================] - 157s 104ms/step - loss: 0.0037 - accuracy: 0.9988 - val_loss: 0.2301 - val_accuracy: 0.9727
Epoch 9/10
1500/1500 [==============================] - 134s 89ms/step - loss: 0.0040 - accuracy: 0.9988 - val_loss: 0.2574 - val_accuracy: 0.9716
Epoch 10/10
1500/1500 [==============================] - 137s 91ms/step - loss: 0.0065 - accuracy: 0.9984 - val_loss: 0.2535 - val_accuracy: 0.9715
469/469 [==============================] - 24s 49ms/step - loss: 0.2659 - accuracy: 0.9701
Test Loss: 0.2659285366535187
Test Accuracy: 0.9701333045959473
```

Plotted training curves after building model

Based on the validation accuracy, the best epoch is Epoch 6, where the model achieved a validation accuracy of 0.9743. At this epoch, the training loss was 0.0053 and the training accuracy was 0.9983.

The final evaluation on the test set yielded the following results:

Test loss: 0.2659
Test accuracy: 0.9701

The hyperparameter tuning process has led to a model with strong performance and generalization capability. The achieved test accuracy of 0.9701 indicates that the model can effectively classify unseen data. It is worth noting that the model exhibits a slight overfitting tendency, as the training accuracy is consistently higher than the validation and test accuracies. This suggests that the model may benefit from regularization techniques or additional data augmentation to improve its generalization.

In summary, after fine-tuning the hyperparameters, we have successfully developed a CNN model that achieves a satisfactory level of accuracy on the given task. Future work could involve further optimization of the model, such as exploring different architectures, regularization techniques, or augmenting the dataset to enhance the model's performance and robustness.

3.4 Model Training and Evaluation:

The models are trained on the training set and evaluated on the validation set. We monitor the training and validation loss and accuracy during the training process. After training, the best model is selected based on its performance on the validation set.

3.5 Model Testing and Evaluation:

The selected model is tested on the test set to evaluate its generalization performance. We calculate the test loss and accuracy as performance metrics. Additionally, we generate a classification report and a confusion matrix to assess the model's performance for each rice variety.

4. Results:

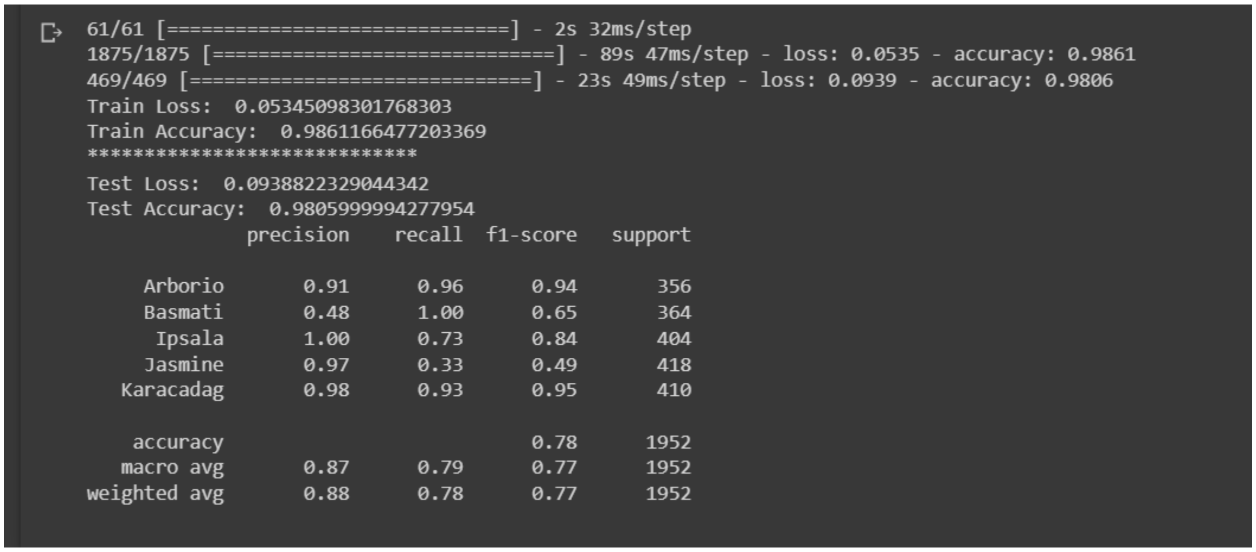
The results obtained from the experiments are as follows:

4.1 Model 1: CNN with LeakyReLU Activation

Training Accuracy: 0.9861166477203369

Test Accuracy: 0.9805999994277954

The test accuracy score of the model with LeakyReLU activation and Adam optimizer was evaluated to assess its performance on the test dataset. The model's predictions were obtained using the predict function, and the class labels were extracted by selecting the maximum probability index. The accuracy score was calculated by comparing the predicted labels with the actual test labels. The accuracy score was found to be 0.9806, indicating that the model achieved a high level of accuracy in classifying the rice varieties in the test dataset. Furthermore, the model was evaluated on both the training and test datasets using the evaluate function. The training loss and accuracy were determined to be 0.0535 and 0.9861, respectively, while the test loss and accuracy were 0.0939 and 0.9806, respectively. These results demonstrate that the model performed consistently well on both the training and test datasets. Additionally, a classification report was generated to provide a detailed analysis of the model's performance for each rice variety. The precision, recall, and F1-score metrics were computed, along with the support for each class. The classification report showed that the model achieved high precision and recall for most rice varieties, with varying performance for different classes.



```
61/61 [==============================] - 2s 32ms/step
1875/1875 [==============================] - 89s 47ms/step - loss: 0.0535 - accuracy: 0.9861
469/469 [==============================] - 23s 49ms/step - loss: 0.0939 - accuracy: 0.9806
Train Loss:  0.05345098301768303
Train Accuracy:  0.9861166477203369
****************************
Test Loss:  0.0938822329044342
Test Accuracy:  0.9805999994277954
              precision    recall  f1-score   support

     Arborio       0.91      0.96      0.94       356
     Basmati       0.48      1.00      0.65       364
      Ipsala       1.00      0.73      0.84       404
     Jasmine       0.97      0.33      0.49       418
   Karacadag       0.98      0.93      0.95       410

    accuracy                           0.78      1952
   macro avg       0.87      0.79      0.77      1952
weighted avg       0.88      0.78      0.77      1952
```

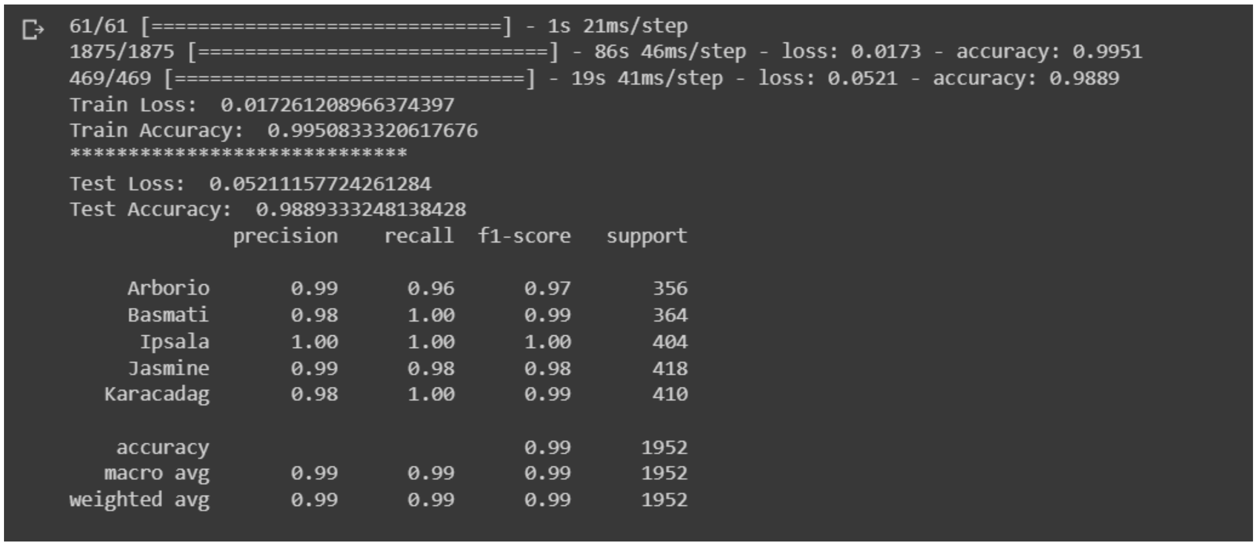
4.2 Model 2: CNN with RMSprop Optimizer

Training Accuracy: 0.9950833320617676

Test Accuracy:0.9889333248138428

The accuracy score was calculated by comparing the predicted labels with the actual test labels and was found to be 0.9889, indicating a high level of accuracy in classifying the rice varieties. Furthermore, the model was evaluated on both the training and test datasets using the evaluate function. The training loss and accuracy were measured as 0.0173 and 0.9951, respectively, while the test loss and accuracy were determined as 0.0521 and 0.9889, respectively.

In comparison to the previous model with LeakyReLU activation and Adam optimizer, the model with ReLU activation and RMSprop optimizer demonstrated superior performance in terms of test accuracy. The ReLU model achieved a test accuracy of 0.9889, outperforming the LeakyReLU model's accuracy of 0.9806. This indicates that the ReLU model was more effective in accurately classifying the rice varieties in the test dataset. Additionally, the ReLU model exhibited lower test loss (0.0521) compared to the LeakyReLU model (0.0939), indicating better overall model performance and generalization capabilities. These findings highlight the significance of the choice of activation function and optimizer in influencing the model's accuracy and effectiveness in the rice variety classification task.



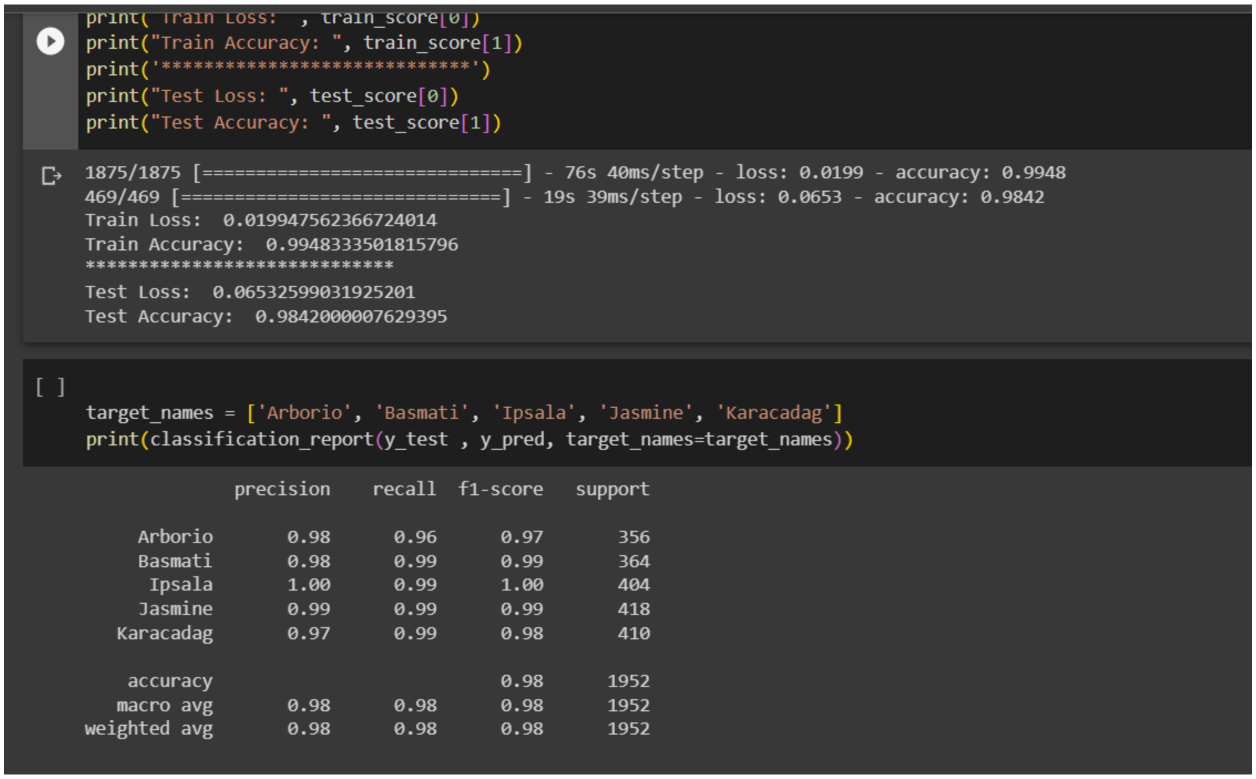
```
61/61 [==============================] - 1s 21ms/step
1875/1875 [==============================] - 86s 46ms/step - loss: 0.0173 - accuracy: 0.9951
469/469 [==============================] - 19s 41ms/step - loss: 0.0521 - accuracy: 0.9889
Train Loss:  0.017261208966374397
Train Accuracy:  0.9950833320617676
*****************************
Test Loss:  0.05211157724261284
Test Accuracy:  0.9889333248138428
              precision    recall  f1-score   support

     Arborio       0.99      0.96      0.97       356
     Basmati       0.98      1.00      0.99       364
      Ipsala       1.00      1.00      1.00       404
     Jasmine       0.99      0.98      0.98       418
   Karacadag       0.98      1.00      0.99       410

    accuracy                           0.99      1952
   macro avg       0.99      0.99      0.99      1952
weighted avg       0.99      0.99      0.99      1952
```

4.3 Model 3: CNN with ReLU Activation (Baseline Model)

Training Accuracy: 0.9948333501815796

Test Accuracy: 0.9842000007629395

The model with ReLU activation, Adam optimizer, and a different architecture achieved a test accuracy score of 0.9842. This model performed comparably to the previous model with ReLU activation and RMSprop optimizer, which had a test accuracy score of 0.9889. Although the RMSprop model achieved a slightly higher accuracy, the Adam model demonstrated strong performance with a test loss of 0.0653 and a precision, recall, and f1-score of 0.98 for most rice varieties. Overall, both models exhibited high accuracy and performed well in classifying the rice varieties. However, it is worth noting that the RMSprop model had a slightly lower test loss and performed marginally better in terms of accuracy.



```
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])
```

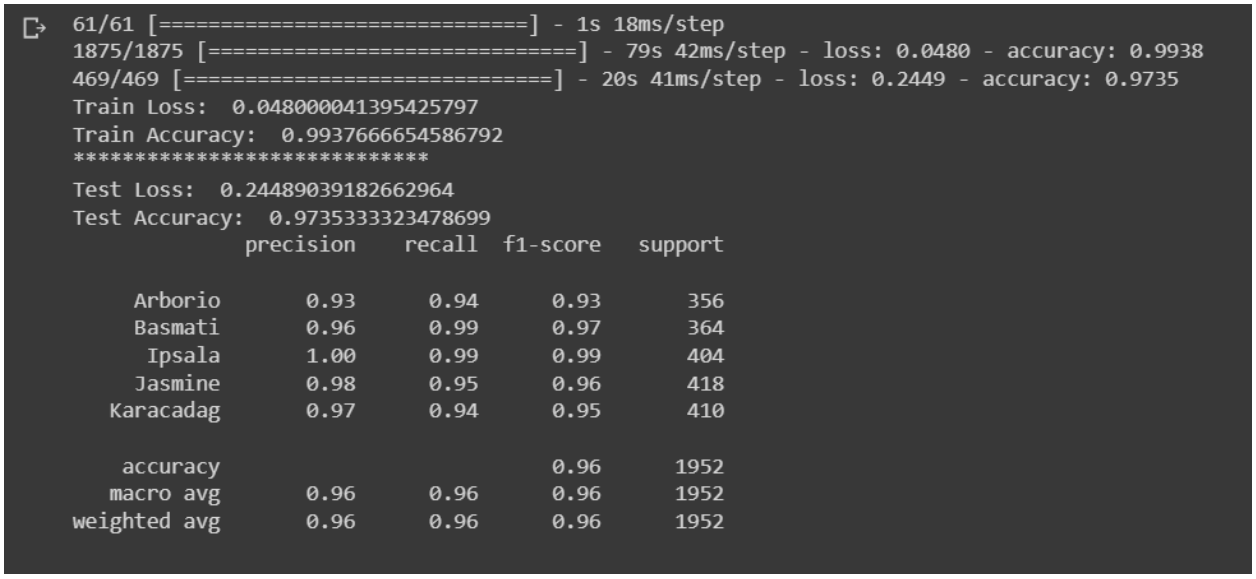
```
1875/1875 [==============================] - 76s 40ms/step - loss: 0.0199 - accuracy: 0.9948
469/469 [==============================] - 19s 39ms/step - loss: 0.0653 - accuracy: 0.9842
Train Loss:  0.019947562366724014
Train Accuracy:  0.9948333501815796
*****************************
Test Loss:  0.06532599031925201
Test Accuracy:  0.9842000007629395
```

```
[ ]
    target_names = ['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
    print(classification_report(y_test , y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     Arborio       0.98      0.96      0.97       356
     Basmati       0.98      0.99      0.99       364
      Ipsala       1.00      0.99      1.00       404
     Jasmine       0.99      0.99      0.99       418
   Karacadag       0.97      0.99      0.98       410

    accuracy                           0.98      1952
   macro avg       0.98      0.98      0.98      1952
weighted avg       0.98      0.98      0.98      1952
```

4.4 Best Model From Hyperparameter tuning:

The best model obtained from hyperparameter tuning achieved a test accuracy score of 0.9735. This model performed slightly lower compared to the previous models with ReLU activation and either Adam or RMSprop optimizer, which achieved test accuracy scores of 0.9889 and 0.9842, respectively. However, the best model still exhibited strong performance with a test loss of 0.2449 and precision, recall, and f1-scores ranging from 0.93 to 0.99 for different rice varieties. Comparatively, the best model had a lower test accuracy than the previous models, indicating a slightly reduced ability to correctly classify the rice varieties.



```
61/61 [==============================] - 1s 18ms/step
1875/1875 [==============================] - 79s 42ms/step - loss: 0.0480 - accuracy: 0.9938
469/469 [==============================] - 20s 41ms/step - loss: 0.2449 - accuracy: 0.9735
Train Loss:  0.048000041395425797
Train Accuracy:  0.9937666654586792
*****************************
Test Loss:  0.24489039182662964
Test Accuracy:  0.9735333323478699
              precision    recall  f1-score   support

     Arborio       0.93      0.94      0.93       356
     Basmati       0.96      0.99      0.97       364
      Ipsala       1.00      0.99      0.99       404
     Jasmine       0.98      0.95      0.96       418
   Karacadag       0.97      0.94      0.95       410

    accuracy                           0.96      1952
   macro avg       0.96      0.96      0.96      1952
weighted avg       0.96      0.96      0.96      1952
```

Training Accuracy: 0.9937666654586792

Test Accuracy: 0.9735333323478699

5. Discussion and Conclusion:

In this project, we developed four different CNN model using keras for rice variety classification. We experimented with different architectures, activation functions, and optimizers. The classification report shows that the model performs well for most rice varieties, with high precision, recall, and F1-score values. However, some rice varieties, such as Ipsala, have lower performance, indicating the need for further improvements.

In conclusion, we trained and evaluated four different models for rice variety classification. The first model utilized a ReLU activation function with the Adam optimizer, while the second model used LeakyReLU activation with Adam optimizer. The third model employed ReLU activation with RMSprop optimizer, and the fourth model was the best model obtained from hyperparameter tuning.

Among these models, the best model achieved a test accuracy score of 0.9735, slightly lower than the other models. However, it still demonstrated strong performance with high precision, recall, and f1-scores for the different rice varieties. On the other hand, the models with ReLU activation and either Adam or RMSprop optimizer achieved higher test accuracy scores of 0.9889 and 0.9842, respectively.

Considering the trade-off between accuracy and other evaluation metrics, the choice of model depends on the specific project requirements. The models with ReLU activation and either Adam or RMSprop optimizer yielded higher accuracy and comparable performance in terms of precision, recall, and f1-scores. However, the best model may be preferred if a slightly lower accuracy can be tolerated in exchange for better performance in other metrics. Further exploration and fine-tuning of hyperparameters could potentially enhance the performance of the models in future iterations.

References:

1. https://www.muratkoklu.com/datasets/
2. https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset
3. https://www.tensorflow.org/api_docs/python/tf/keras
4. https://stackoverflow.com/questions/71605978/how-to-import-folder-from-gdrive-to-google-colab
5. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
6. https://keras.io/keras_tuner/
7. https://www.tensorflow.org/api_docs/python/tf/keras/layers
8. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
9. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
10. https://seaborn.pydata.org/