

Rice Image Classification using Hadoop environment.

Instructor: Hema Seshadri

Table of Contents

[Introduction 3](#_Toc138680857)

[Hadoop Ecosystem 3](#_Toc138680858)

[Code Walkthrough 5](#_Toc138680859)

[Modeling 7](#_Toc138680860)

[Model Evaluation 9](#_Toc138680861)

[Conclusion 11](#_Toc138680862)

# Introduction

This report aims to leverage Bigdata framework in order to solve Image classification problem. The dataset used for this process is the Rice dataset. The dataset has 5 categories and each category has around 15,000 sample images. The dataset also has 12 morphological, 4 shape, and 90 color features. This data focuses on varieties such as Arborio, Basmati, Ipsala, Jasmine, and Karacadag rice.

Hadoop ecosystem was designed in order to read data from HDFS into Spark where the images are processed and result are stored. For this project we have used PySpark (Spark extension that allows python integration). The data has been trained using CNN model.

# Hadoop Ecosystem

The Hadoop ecosystem is a collection of open-source software frameworks and tools designed to process and analyze large datasets in a distributed computing environment. It provides scalable, reliable, and cost-effective solutions for big data processing. Two key components of the Hadoop ecosystem are the Hadoop Distributed File System (HDFS) and PySpark.

HDFS, short for Hadoop Distributed File System, is a distributed file system that allows data to be stored and accessed across multiple nodes in a Hadoop cluster. It provides fault tolerance, high throughput, and the ability to handle large datasets by splitting them into smaller blocks and distributing them across the cluster. HDFS follows a master-slave architecture, where the NameNode serves as the master and manages the file system's metadata, while the DataNodes act as slaves and store the actual data blocks.

PySpark, on the other hand, is the Python library for Apache Spark, an open-source, cluster-computing framework designed for big data processing and analytics. Spark provides a unified data processing model that supports various data processing tasks, including batch processing, streaming, machine learning, and graph processing. PySpark enables developers to write Spark applications using Python, making it a popular choice for data scientists and Python developers.

When using HDFS and PySpark together, you can leverage the power of distributed storage and processing. HDFS acts as the underlying storage layer for your data, allowing you to store and retrieve large datasets efficiently. PySpark, with its Python API, provides a high-level interface to interact with the data stored in HDFS and perform complex data processing tasks.

By combining HDFS and PySpark, you can build data pipelines that read data from HDFS, process it using Spark's powerful distributed computing capabilities, and store the results back in HDFS or any other supported data storage system. This combination enables you to handle large-scale data processing tasks in a distributed and scalable manner, taking advantage of the fault tolerance and parallel processing capabilities offered by the Hadoop ecosystem.

PySpark

(Image Processing)

HDFS

(Storage System)

For this project we have installed PySpark in our local system and integrated it with Anaconda to enable Spark in Jupyter Notebook. While installing the system it is very important to take care of the version of Hadoop, Spark, Python, and Keras. Make sure that you have a seamless compatibility between them.

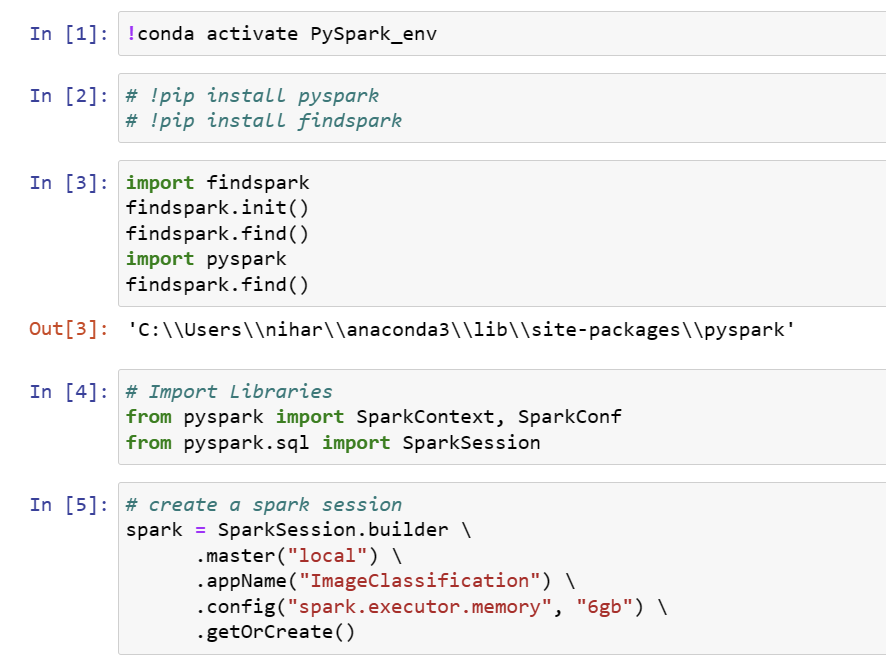
In this project we have used the below configuration.

1. Hadoop version 3.2
2. Spark version 3.2.4
3. Python version 3.9
4. TensorFlow version 2.12.0
5. Keras version 2.12.0

If you require more help on how to install and integrate PySpark in your local windows machine, you can follow this link <https://blog.datamics.com/how-to-install-pyspark-on-windows-faf7ac293ecf> . The blog in this link has provided step-by-step integration walkthrough which we have used for this project.

# Code Walkthrough

Once the integration is in place and you are able to load the notebook using spark, run the blow code to see if you are able to access Spark or not.



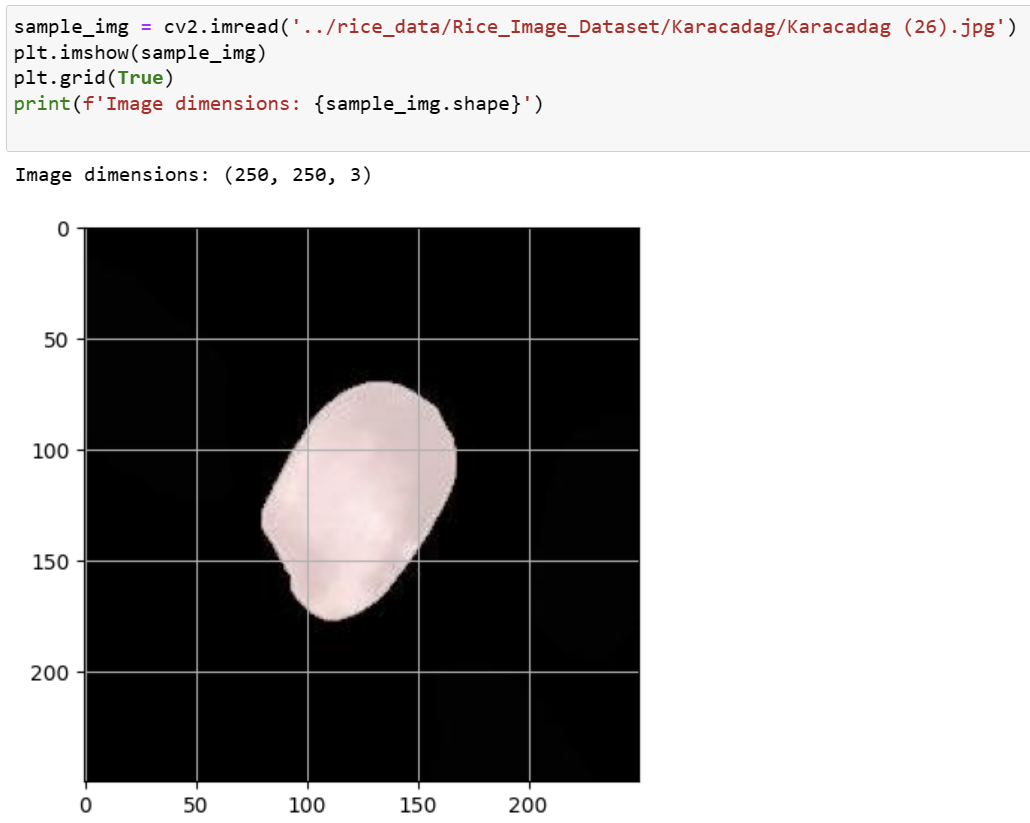
If you are able to run the above code successfully, you can be assured that your integration was done properly.

We have deployed HDFS in my local system and has allocated 64GB of space in our local machine and hence the file path is directing to our local machine.

Once the Spark Session has been activated, we can then load the images and label them using the code below.



We can then view a sample image using matplotlib:



After we make sure that the images are read properly, we can then more on to creating a model and train it using the train dataset.

# Modeling

Modeling is the stage where we will tell our system how to distinguish between the rice types. To do so we have used CNN algorithm. A Convolutional Neural Network (CNN) is a type of deep learning model that is widely used for image recognition and computer vision tasks. CNNs are inspired by the visual processing mechanism of the human brain and are designed to automatically learn and extract meaningful features from images.

The fundamental building blocks of a CNN are convolutional layers, which perform convolutions on input images using a set of learnable filters. These filters capture different features such as edges, textures, or shapes. The convolutional layers help in detecting and recognizing patterns by convolving the filters across the entire image, producing feature maps that highlight the presence of specific features.

CNNs also utilize pooling layers to down-sample the spatial dimensions of the feature maps. Pooling helps in reducing the computational complexity and extracting the most salient features. Common pooling techniques include max pooling, which selects the maximum value within each pooling region, and average pooling, which takes the average value.

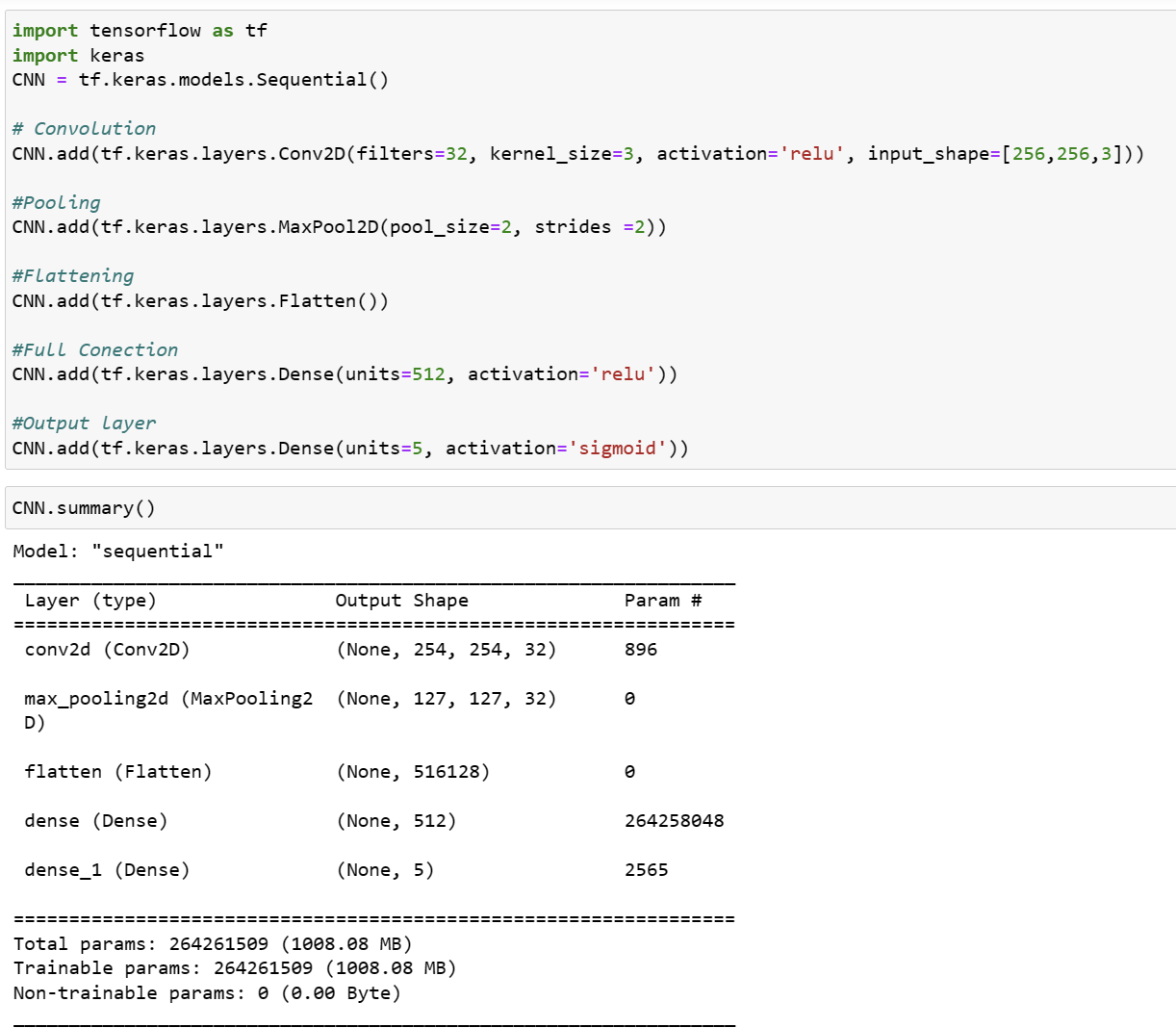
The output of the convolutional and pooling layers is then passed through one or more fully connected layers. These layers perform classification or regression tasks based on the extracted features. Activation functions like ReLU (Rectified Linear Unit) are typically used between the layers to introduce non-linearity and make the model capable of learning complex relationships.

During training, CNNs employ a technique called backpropagation to update the weights and biases of the filters and fully connected layers based on the computed error. The model learns to optimize its parameters by minimizing a loss function, typically through techniques like gradient descent.

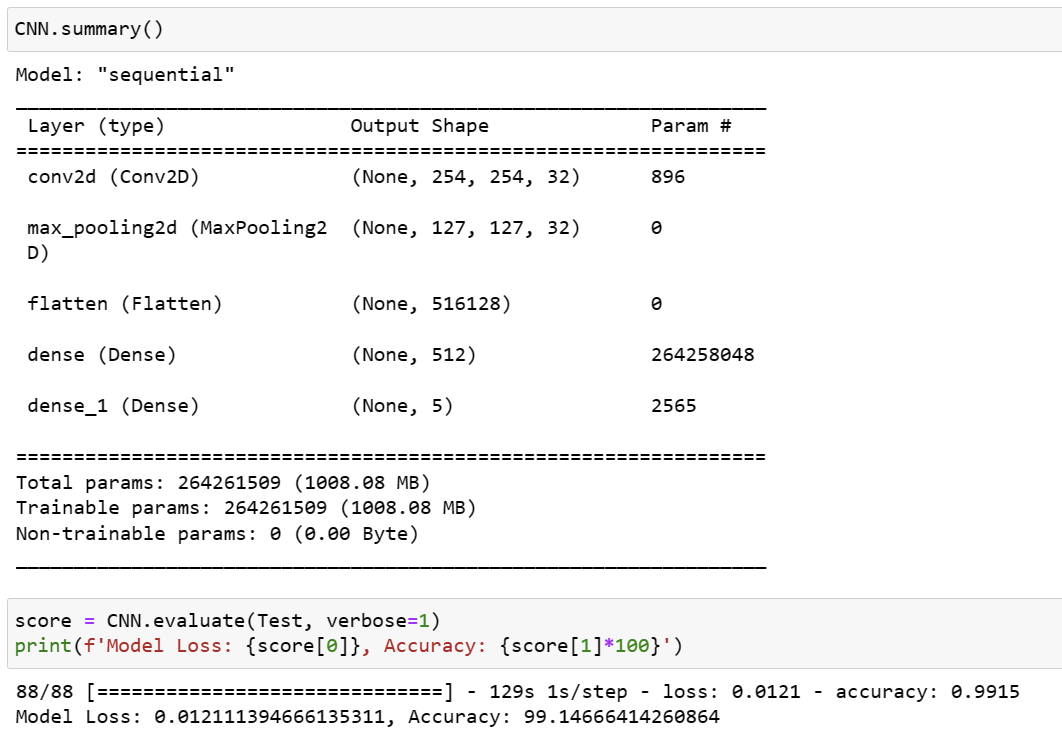
CNNs have achieved remarkable success in various computer vision tasks, including image classification, object detection, semantic segmentation, and image generation. They have outperformed traditional machine learning approaches due to their ability to automatically learn hierarchical representations directly from raw pixel data.

With the availability of large annotated datasets and advancements in hardware, CNNs have become increasingly powerful and are widely used in various applications, including autonomous vehicles, medical imaging, facial recognition, and more. Their ability to automatically learn and extract features from images makes them a crucial tool for solving complex visual problems.

We have created a CNN model using the code below:



Once the model has been created, we need to test the data on our validation set to see if the results are satisfactory.

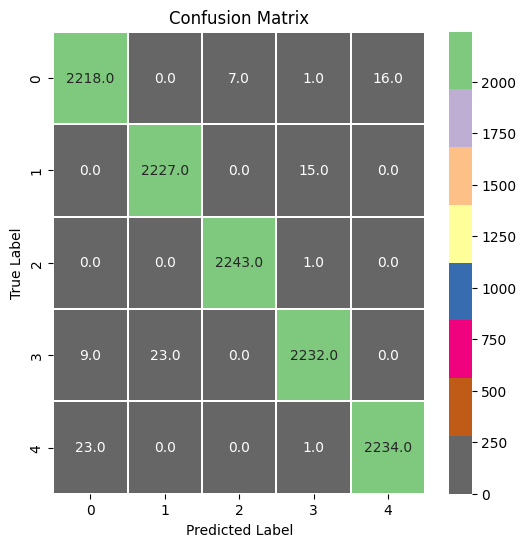


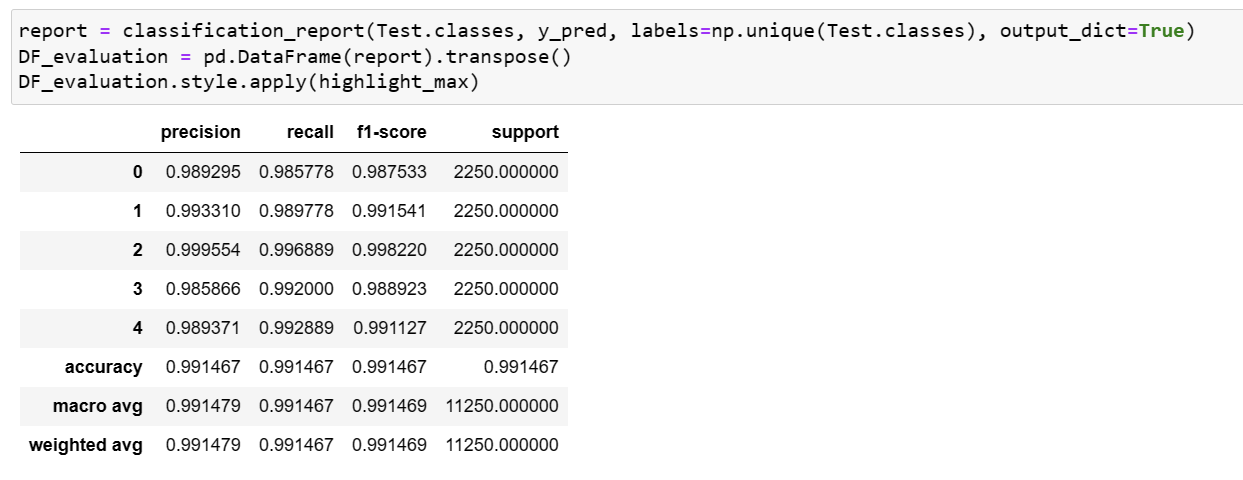
The model was fine tuned based on the accuracy but it is not the only Metrix we need to consider while evaluating the model. But based on the results we are getting 99% of accuracy from our model.

# Model Evaluation

For the model evaluation we will looking at the confusion Metrix and then at other evaluation parameters to evaluate the overall performance of the model. The confusion metrix is as shown below:

It can be observed that our model sometimes get confused between label 0 and label 4, and label 1 and label 3. Apart from that all the other labels are getting recognized and classified properly and the result is satisfactory.





Based on the Precision, Recall and F1 score, it can be observed that the model is scoring close to 99% for each parameter. This means that our model is performing good across multiple parameters and can conclude that it is a good model and can be deployed for rice type classification in the production environment.

# Conclusion

In conclusion, this report aimed to leverage the Bigdata framework to solve the image classification problem using the Rice dataset. The Hadoop ecosystem, specifically HDFS and PySpark, was utilized to read and process the data. PySpark, an extension of Apache Spark with Python integration, provided a high-level interface for data processing. The dataset was trained using a Convolutional Neural Network (CNN) model, a deep learning architecture widely used for image recognition tasks.

The Hadoop ecosystem, with its distributed file system (HDFS) and Spark's distributed computing capabilities (PySpark), enabled efficient handling of large-scale data processing tasks. By combining these components, data pipelines were built to read data from HDFS, process it using Spark's powerful distributed processing capabilities, and store the results back in HDFS.

The CNN model, designed for image recognition, utilized convolutional layers to extract features from images, followed by pooling layers for downsampling and fully connected layers for classification. The model was trained using backpropagation, optimizing its parameters based on a loss function. CNNs have proven effective in various computer vision tasks, thanks to their ability to automatically learn hierarchical representations from raw pixel data.

The model evaluation included analyzing the confusion matrix and other evaluation metrics such as precision, recall, and F1 score. The model achieved a high accuracy rate of approximately 99% and demonstrated good performance across multiple evaluation parameters. This indicates that the model is reliable and suitable for deployment in a production environment for rice type classification.

Overall, the utilization of the Hadoop ecosystem, including HDFS and PySpark, along with the CNN model, showcased the potential of leveraging big data frameworks for image classification tasks, enabling efficient processing and accurate results.

# References

Chaturvedi, N. (2023, May 24). *Implemented neural networks projects*. Medium. https://medium.com/coders-mojo/implemented-neural-networks-projects-d25a6476d72b

Dertat, A. (2017, November 13). *Applied deep learning - part 4: Convolutional Neural Networks*. Medium. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Salman Ahmad Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, & Ahmad, S. (2022, July 1). *Confusion matrix-based modularity induction into pretrained CNN*. Multimedia Tools and Applications. https://dl.acm.org/doi/abs/10.1007/s11042-022-12331-2

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. https://doi.org/10.1109/msst.2010.5496972

*Spark Streaming Programming Guide*. Spark Streaming - Spark 3.4.1 Documentation. (n.d.). https://spark.apache.org/docs/latest/streaming-programming-guide.html

Team, D. (2019, April 26). *Hadoop ecosystem and their components - a complete tutorial*. DataFlair. https://data-flair.training/blogs/hadoop-ecosystem-components/

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018, June 22). *Convolutional Neural Networks: An overview and application in radiology - insights into imaging*. SpringerOpen. https://doi.org/10.1007/s13244-018-0639-9