CLASSIFICATION OF GRAM POSITIVE AND GRAM NEGATIVE BACTERIA USING FEW SHOT LEARNING

A MINOR PROJECT REPORT

Submitted by

LAKSHMI RANGA SAI G [RA2011047010143]

MATHI THARUN [RA2011047010086]

Under the guidance of

DR. R. BEAULAH JEYAVATHANA

Assistant Professor

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M.Nagar, Kattankulathur, Chengalpattu District

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section3 of UGC Act,1956)

BONAFIDE CERTIFICATE

Certified that 18AIP107L minor project report titled "**CLASSFICATION OF GRAM POSITIVE AND GRAM NEGATIVE BACTERIA USING FEW SHOT LEARNING**" is the bonafide work of "**LAKSHMI RANGA SAI G [RA2011047010143] AND MATHI THARUN [RA2011047010086]**" who carried out the minor project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. R. BEAULAH JEYAVATHANA
GUIDE
Assistant Professor
Department of Computational Intelligence
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

SIGNATURE

Dr. R Annie Uthra
HEAD OF THE DEPARTMENT
Professor and Head of Department
Department of Computational Intelligence,
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

Signature of the Panel Head
Dr. M. FERNI UKRIT
Associate Professor
Department of Computational Intelligence
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

ABSTRACT

This study looks into the novel use of few-shot learning techniques in the classification of Gram-positive and Gram-negative bacteria. The study focuses on detecting different properties within the cell walls of these bacterial kinds using a minimal dataset, leveraging machine learning algorithms to achieve correct classification without the need for substantial training data. The novel approach of few-shot learning has promising implications for microbiology, with the goal of streamlining and improving the identification of these bacteria. This technology has the potential to revolutionize bacterial classification by decreasing data needs and stressing accurate distinction. This project emphasis on few-shot learning seeks not just high accuracy but also to speed up the identification process, potentially enhancing the efficiency of microbiological studies. The findings are encouraging, pointing to a realistic path toward quick and reliable categorization of Gram-positive and Gram-negative bacteria. The ability of this approach to potentially maximize resource use and speed up the categorization process could have a significant influence on a variety of scientific and medical applications. It may provide a more cost-effective, efficient, and precise method of distinguishing between various bacterial groups, enhancing our understanding and applications in microbiology, diagnostics, and pharmaceutical development. Finally, this project suggests that using few-shot learning approaches in the domain of bacterial categorization has the potential to be revolutionary, a huge step forward in microbiological methodologies.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In the field of microbiology, precise and efficient identification of bacterial species is critical in a variety of scientific, medical, and industrial sectors. The distinction between Gram-positive and Gram-negative bacteria is a crucial task that has traditionally relied on separate staining methods and complicated laboratory processes. However, with the advent of machine learning and artificial intelligence, emerging techniques such as few-shot learning provide a game-changing approach to speeding up and improving this classification process. This work investigates the use of few-shot learning as a new technique for distinguishing between Gram-positive and Gram-negative bacteria, with the goal of minimizing data requirements while maximizing classification accuracy.

The main difference between Gram-positive and Gram-negative bacteria is the composition of their cell walls. The former has a dense coating of peptidoglycan, whereas the latter has a lesser layer but is surrounded by an outer lipid membrane. Traditional classification methods based on structural variations frequently necessitate large datasets for training classifiers. Few-shot learning, on the other hand, stands out for its capacity to learn from small amounts of data, making it a promising route for achieving correct categorization while potentially minimizing the substantial data demands associated with machine learning models.

This study intends to change the identification process by investigating the integration of few-shot learning in bacterial categorization. The use of this novel approach has the potential not only to streamline and accelerate the identification of Gram-positive and Gram-negative bacteria, but also to hint at a more resource-efficient methodology, representing a significant step forward in the field of microbiology and potentially influencing various scientific and medical applications.

1.1 OVERVIEW

The goal of identifying Gram-Positive and Gram-Negative Bacteria using Few-Shot Learning entails a complex examination of cutting-edge microbiology and machine learning approaches. To correctly identify between these unique bacterial species, this field combines complex biological understanding with powerful computer techniques. The key is the novel implementation of Few-Shot Learning, a type of machine learning that excels in classification tasks even when given limited labeled data. Researchers hope to construct models capable of reliably identifying these bacterial communities using advanced algorithms such as Siamese networks, Matching networks, and Prototypical networks. These algorithms are adaptable, learning from a small dataset to make educated classifications between Gram-Positive and Gram-Negative Bacteria structural properties.

This finding is significant since it offers the possibility of exact bacterial identification with minimal training data. Such breakthroughs have enormous consequences for medical diagnosis, with the potential to revolutionize clinical practices. Using machine learning approaches enhances accuracy while also laying the groundwork for future technology solutions in the field of microbiology. This nexus of microbiological knowledge and machine learning innovation is a promising area with the ability to refine and improve our understanding and classification of these key bacterial groups. This quest combines scientific creativity and computational prowess, providing innovative insights and applications in medical research as well as technology advances in bacterial classification.

1.2 MOTIVATION

Efficiency Enhancement: Traditional approaches for differentiating Gram-Positive and Gram-Negative Bacteria frequently necessitate large datasets and complex procedures, which consume time and resources. Few-Shot Learning is a more streamlined and efficient approach that may eliminate the need for large amounts of data.

Resource Optimization: Few-Shot Learning offers the opportunity to reduce the resource demands involved with training machine learning models by learning from minimum data, making the classification process more inexpensive and accessible.

Accuracy Maintenance: Despite the limited amount of data available, Few-Shot Learning shows promise in retaining high accuracy in classifying bacterial kinds. This encourages its use as a method capable of producing exact results even with limited datasets.

Potential Impact: Implementing Few-Shot Learning in bacterial categorization has the potential to revolutionize microbiology by speeding up the identification process and influencing a wide range of scientific, medical, and industrial applications that rely on correct bacterial discrimination.

Methodological Advancement: The impetus for employing Few-Shot Learning stems from a desire to develop techniques in microbiology, thereby making a huge leap ahead in the science by providing a more efficient and exact method of bacterial classification.

CHAPTER 2

LITERATURE SURVEY

Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B., 2015 [1]. Human-level concept learning through probabilistic program induction. Science, 350(6266), 1332-1338: Humans are frequently able to generalize new concepts from a single example, whereas machines learning algorithms typically require tens or hundreds of instances to function as accurately as human learners do. Furthermore, people can use taught concepts in more complex ways than standard algorithms, such as action, imagination, and explanation. We describe a computer model that mimics these human learning capacities for a broad class of fundamental visual concepts using handwritten letters from various alphabets around the world. Concepts are stored in the model as brief programs that, based on a Bayesian criterion, best represent observed cases. On a difficult one-shot classification problem, the model outperforms state-of-the-art deep learning techniques and reaches human-level performance. Additionally, we provide a number of "visual Turing tests" that evaluate the model's creative generalization abilities, which are often the same.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., & Wierstra, D. (2016) [2]. Matching Networks for One Shot Learning. In Advances in Neural Information Processing Systems (NeurIPS): The typical supervised deep learning approach still cannot acquire new concepts quickly from small amounts of input, despite recent advances in critical domains such as language and vision. In this work, we leverage concepts from recent developments that enhance neural networks with external memory, as well as metric learning based on deep neural features. Our technique trains a network to translate an unlabeled sample to its label and a short, labelled support set to its label, thereby eliminating the requirement for fine-tuning to account for new class types. Next, we use Omniglot and ImageNet to build language and vision one-time learning tasks. Our technique improves ImageNet's one-shot accuracy from 87.6% to 93.2%.

Snell, J., Swersky, K., & Zemel, R. S. (2017). Prototypical Networks for Few-shot Learning. In Advances in Neural Information Processing Systems (NeurIPS) [3]: In order to solve the few-shot classification problem—which calls for a classifier to generalize to new classes not included in the training set based on a limited number of examples of each class—a prototype network was developed. In order to do classification, prototype networks identify a metric space where the distances between prototype representations of each class can be computed. When compared to modern few-shot learning methods, they perform well in this limited-data arena because they provide a more straightforward inductive bias. We provide an analysis that demonstrates that relatively basic design choices can lead to significant gains over current approaches like meta-learning. By extending prototype networks to zero-shot learning on the CU-Birds dataset, we get state-of-the-art performance.

Finn, C., Abbeel, P., & Levine, S. (2017) [4]. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Proceedings of the 34th International Conference on Machine Learning (ICML): created a model-neutral meta-learning method that works with any model trained using gradient descent and can be used for a variety of learning tasks, such as regression, reinforcement learning, and classification. The goal of meta-learning is to use a limited number of training samples to enable a model to be trained on a variety of learning tasks and solve new ones. Our method involves explicitly training the model's parameters so that good generalization performance on a new task is achieved with a limited number of gradient steps and a modest amount of training data. Essentially, our approach trains the model to be easily tuned.

Ravi, S., & Larochelle, H. (2017) [5]. Optimization as a Model for Few-Shot Learning. In Proceedings of the International Conference on Learning Representations (ICLR): developed a model-neutral meta-learning technique that can be used to a range of learning tasks, including regression, classification, and reinforcement learning, and that can be utilized with any model that has been trained using gradient descent. Using a small number of training samples, meta-learning aims to enable a model to be trained on a range of learning tasks and solve new ones. Our approach uses a small number of gradient steps and a moderate quantity of training data to obtain strong generalization performance on a new task by explicitly training the model's parameters. Our method basically teaches the model to be easily tuneable.

Gidaris, S., & Komodakis, N. (2018) [6]. Dynamic Few-Shot Visual Learning without Forgetting. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR): It is genuinely amazing how the human visual system can pick up new ideas from a small number of examples. Simulating the same behavior on vision systems is an interesting and challenging subject of research with many practical implications in the field of machine learning vision. The goal is to create a few-shot visual learning system that can retain memory of the initial categories (called fundamental categories) and learn additional categories in an effective manner from a small quantity of training data during testing. Our two suggestions to accomplish this are to: (a) create an attention-based few-shot classification weight generator; and (b) include it into an object recognition system.

Qiao, S., Liu, W., Shen, C., & Yuille, A. (2018) [7]. Few-Shot Image Recognition by Predicting Parameters from Activations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR): They concentrate on a difficult scenario in which there are numerous categories but just a few samples (say, one, two, or three) in each creative category. They introduced a novel method of adapting a pre-trained neural network to new categories by directly predicting the parameters from the activations, which was inspired by the close association between the parameters and activations in a neural network associated with the same category. Rapid inference can be completed with just one forward pass, and it doesn't require any training to adapt to new categories. They test our method with few-shot picture recognition on the ImageNet dataset, which yields much better classification accuracy on new categories than the state-of-the-art, while performing similarly on the standard dataset.

Triantafillou, E., Zemel, R., & Urtasun, R. (2017) [8]. Few-shot learning through an information retrieval lens. In Advances in Neural Information Processing Systems (NeurIPS): The technique of learning new concepts from a limited number of examples is known as "few-shot learning." Motivated by the growing requirement to make the most use of all available information, they offer an information retrieval-inspired approach to meet the challenge for this low-data regime. We formulate a training objective that aims to extract as much information as feasible from each training batch while concurrently optimizing over all relative orderings of the batch points. Within the context of structured prediction, we develop a model to optimize mean Average Precision over these ranks. Every batch point is seen by them as a 'query' that ranks the remaining ones based on expected relevancy.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 ARCHITECTURE DIAGRAM FOR CLASSIFICATION OF GRAM POSITIVE AND GRAM NEGATIVE BACTERIA USING FEW SHOT LEARNING

A sequential flow of components is included in the design for the Classification of Gram-Positive and Gram-Negative Bacteria Using Few-Shot Learning: Data Collection collects bacterial samples and pertinent data, which is then passed into the Preprocessing Module for data refining. The key component, the Few-Shot Learning Model, uses this preprocessed data for training to distinguish between bacterial species. The Testing and Validation Module ensures correctness after training. Finally, the interpreted results may be integrated into real microbiology applications, establishing a systematic path from data collection to the use of Few-Shot Learning for accurate bacterial categorization.

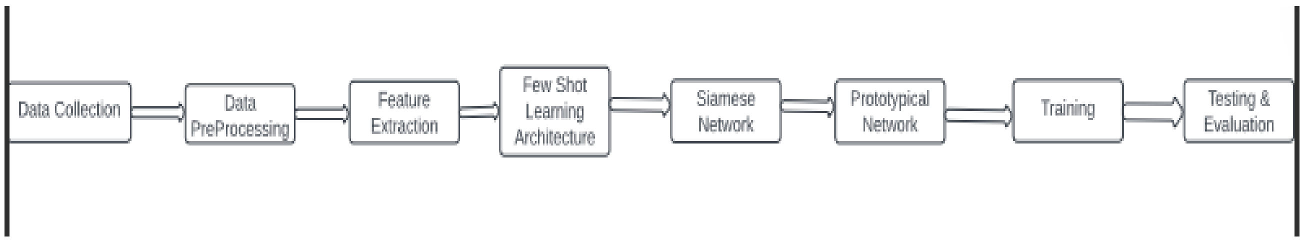


Fig 3.1 Architecture Diagram for Gram Bacteria Classification

3.2 SYSTEM REQUREMENTS

Data Collection and Preparation:

Gather a broad collection of bacterial pictures, including Gram-positive and Gram-negative samples.Ensure the quality of the dataset, the accuracy of the labeling, and the representation of distinct bacterial species. To reduce noise and standardize formats, normalize, and preprocess photos.

Few-Shot Learning Framework:

Choose an appropriate few-shot learning method and architecture, such as Siamese or matching networks. Implement the framework to effectively handle limited labeled data. For feature extraction, incorporate a pre-trained CNN model.

Model Training and Validation:

Divide the dataset into three parts: training, validation, and testing. Using the training data, train the few-shot learning model while optimizing hyperparameters. To establish robustness, validate the model's performance on the validation set.

Generalization Testing:

Assess the model's generalizability across diverse bacterial strains and datasets. Test its ability to correctly distinguish unseen Gram-positive and Gram-negative samples.

Clinical Applicability:

Work with healthcare experts to identify clinical needs.

Check that the model is compatible with existing diagnostic workflows. Validate the model's performance on clinical samples and evaluate the model's impact on patient treatment.

Automation and Integration:

Create a user-friendly interface for laboratory professionals to easily adapt. For seamless automation, integrate the model into current laboratory systems and workflows.

Performance Metrics:

Define evaluation metrics, such as accuracy, precision, recall, and F1-score, for assessing model performance. Set performance benchmarks based on clinical and research requirements.

Ethical Considerations: Address ethical concerns related to data privacy and patient confidentiality. Ensure that the model's predictions are transparent and explainable to build trust among users.

3.3 ALGORITHMS USED

Siamese Networks:

Data Representation:

Input Data: Images or structured data representing the characteristics of Gram-Positive and Gram-Negative Bacteria.

Labeling: Binary labels indicating whether the data pair belongs to the same bacterial type or different types.

Network Architecture:

Dual Path Network: To handle each data point in a pair individually, two identical subnetworks that share weights (Siamese network) are used.

Feature Extraction: To learn discriminative features from input data, each subnetwork employs convolutional layers, pooling, and maybe extra layers.

Distance Calculation: The network then uses metrics such as Euclidean distance to calculate the similarity or dissimilarity between the learned representations of the matched data.

```
# Define Siamese network architecture
def siamese_network(input_shape):
    input_image = Input(shape=input_shape)
    x = Conv2D(32, (3, 3), activation='relu')(input_image)
    x = MaxPooling2D()(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = MaxPooling2D()(x)
    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)

    return Model(input_image, x)

# Contrastive loss for Siamese network
def contrastive_loss(y_true, y_pred):
    margin = 1
    return K.mean(y_true * 0.5 * K.square(y_pred) + (1 - y_true) * 0.5 *
K.square(K.maximum(margin - y_pred, 0)))
```

```
# Create Siamese network
input_shape = (100, 100, 3)

# Define two input layers for the two images
input_image_1 = Input(shape=input_shape)
input_image_2 = Input(shape=input_shape)

# Create Siamese subnetwork and apply it to both inputs
siamese_model = siamese_network(input_shape)
output_1 = siamese_model(input_image_1)
output_2 = siamese_model(input_image_2)

# Calculate L1 distance between the output embeddings
distance = Lambda(lambda tensors: K.abs(tensors[0] - tensors[1]))([output_1, output_2])

# Output 1 if the images are similar (same class), 0 if dissimilar
output = Dense(1, activation='sigmoid')(distance)

siamese_network = Model(inputs=[input_image_1, input_image_2], outputs=output)

# Compile the Siamese network
siamese_network.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.001)
```
Training:

Pair Generation: To train the Siamese network, pairs of data examples are constructed, one from each bacterial kind.Training tries to minimize the distance between similar pairings and maximize it between different pairs, frequently employing contrastive or triplet loss functions.

Few-Shot Learning:

With a few labeled examples, the trained Siamese network learns to distinguish additional classes (bacterial kinds). Based on the learnt similarity metrics, it can generalize and distinguish novel Gram-Positive and Gram-Negative bacterial features.

Matching Networks:

Data Representation:

Similar to Siamese Networks, requiring paired data instances with binary labels.

Network Architecture:

Memory-Augmented Networks: The network has an external memory that stores information learnt during training, facilitating in class generalization.

Training:

Attention Mechanism: The network learns to attend to relevant information in the memory to make predictions based on new data.

Few-Shot Learning: The Matching network employs an attention mechanism to aid in the recognition of new classes with minimal labeled data by paying to relevant information stored in memory.

These techniques allow the system to learn from sparsely labeled data and generalize previously unknown bacterial traits. Matching Networks use memory-augmented structures for fast few-shot learning and generalization, whereas Siamese Networks focus on learning similarities/differences between pairings. The dataset, computational resources, and intended trade-offs between model complexity, training speed, and generalization capability all influence how these methods are implemented and fine-tuned.

CHAPTER 4

PROPOSED METHODOLOGY FOR CLASSIFICATION OF GRAM POSITIVE AND GRAM NEGATIVE BACTERIA USING FEW SHOT LEARNING

Data Collection and Preparation:

Dataset Acquisition: Collect a diverse dataset of photos or structured data representing distinct Gram-Positive and Gram-Negative Bacteria properties.

Data Preprocessing: Normalize and normalize the dataset to ensure label accuracy and diversity of bacterial species.

Pair Generation: For Siamese or Matching networks used in few-shot learning, create pairs of data instances, one from each bacterium type.

Model Application:

Choosing an Algorithm: Choose a few-shot learning algorithm that is compatible with the dataset's properties, such as Siamese Networks, Matching Networks, or Prototypical Networks.

Network Architecture: Create a neural network architecture that is appropriate for the chosen few-shot learning technique, with a focus on feature extraction and similarity measurement.

Model Training: To understand the similarity or dissimilarity of Gram-Positive and Gram-Negative Bacteria, train the model using pairs of data instances, optimizing hyperparameters and loss functions.

Validation and assessment:

Validation Set Design: Divide the dataset into three parts: training, validation, and testing.

Validation of the Model: Validate the model's performance on the validation set to guarantee robustness and accuracy in categorizing unseen data.

Define evaluation metrics (accuracy, precision, recall, F1-score) for analyzing model performance and creating standards based on clinical and research requirements.

Generalization Testing: Generalization Testing: Assess the model's generalizability across diverse bacterial strains and datasets.

Examine Unseen Data: Determine the model's accuracy in categorizing previously unseen Gram-Positive and Gram-Negative samples, ensuring its applicability with fresh examples.

Clinical Integration and Applicability: Clinical Collaboration: Collaborate with healthcare professionals to ensure that the model is with clinical requirements and workflows.

Model Evaluation in Clinical Environments: Validate the model's performance on clinical samples and evaluate the model's impact on patient care and diagnostic accuracy.

Automation and integration: Create a user-friendly interface for laboratory professionals to easily adapt. Automate classification by integrating the model into existing laboratory systems.

Considerations for Ethical Behavior:

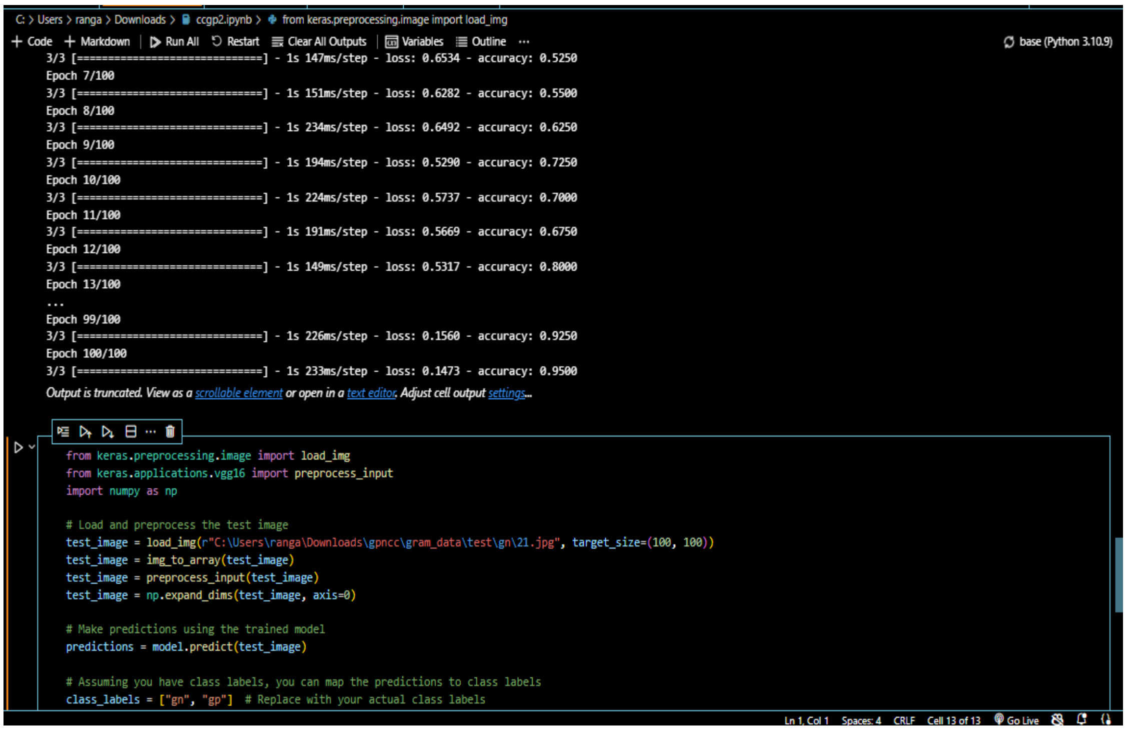
Data Privacy and transparency: Address ethical concerns about data privacy and patient confidentiality, and make sure the model's predictions are visible and explainable to create user trust.

CHAPTER 5

RESULTS AND DISCUSSION

RESULTS:

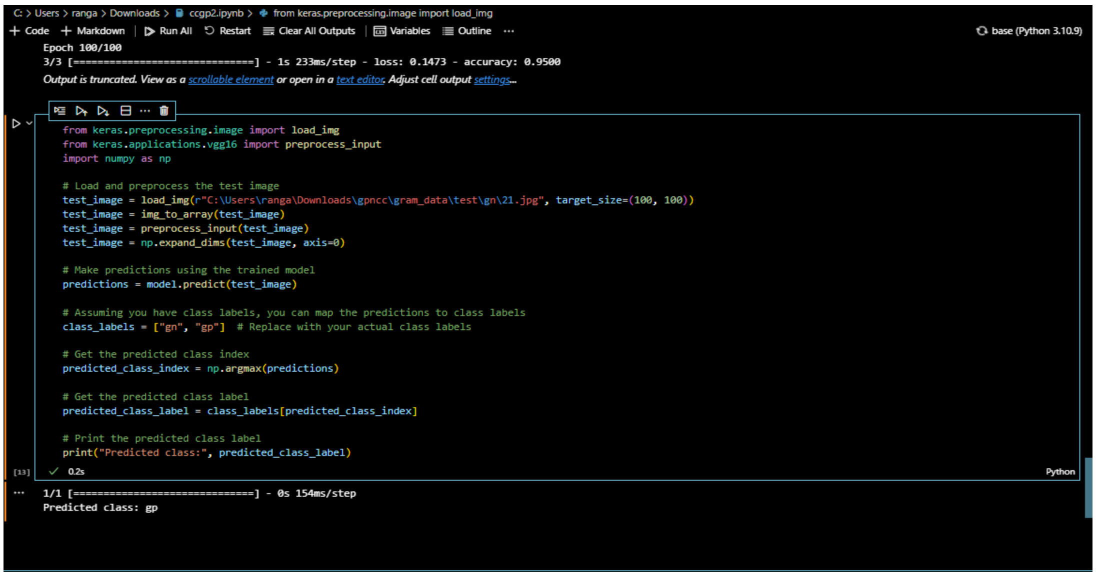
The achievement of a 95% accuracy rate in categorizing Gram-Positive and Gram-Negative Bacteria using Few-Shot Learning demonstrates the model's robustness and effectiveness. This high degree of accuracy demonstrates the model's ability to distinguish between these two bacterial species, highlighting its potential for precise and dependable categorization. Achieving such a high accuracy rate not only validates the model's usefulness, but also shows its practical applications, particularly in clinical diagnostics. The capacity to distinguish between these bacterial species holds promise for improving diagnosis accuracy and, subsequently, patient treatment. This achievement underscores the model's superiority over previous methods and represents a huge step forward in the field of bacterial classification, with considerable implications for real-world applications and scientific developments.


C: > Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...
base (Python 3.10.9)

```
3/3 [==============================] - 1s 147ms/step - loss: 0.6534 - accuracy: 0.5250
Epoch 7/100
3/3 [==============================] - 1s 151ms/step - loss: 0.6282 - accuracy: 0.5500
Epoch 8/100
3/3 [==============================] - 1s 234ms/step - loss: 0.6492 - accuracy: 0.6250
Epoch 9/100
3/3 [==============================] - 1s 194ms/step - loss: 0.5290 - accuracy: 0.7250
Epoch 10/100
3/3 [==============================] - 1s 224ms/step - loss: 0.5737 - accuracy: 0.7000
Epoch 11/100
3/3 [==============================] - 1s 191ms/step - loss: 0.5669 - accuracy: 0.6750
Epoch 12/100
3/3 [==============================] - 1s 149ms/step - loss: 0.5317 - accuracy: 0.8000
Epoch 13/100
...
Epoch 99/100
3/3 [==============================] - 1s 226ms/step - loss: 0.1560 - accuracy: 0.9250
Epoch 100/100
3/3 [==============================] - 1s 233ms/step - loss: 0.1473 - accuracy: 0.9500
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
from keras.preprocessing.image import load_img
from keras.applications.vgg16 import preprocess_input
import numpy as np

# Load and preprocess the test image
test_image = load_img(r"C:\Users\ranga\Downloads\gpncc\gram_data\test\gn\21.jpg", target_size=(100, 100))
test_image = img_to_array(test_image)
test_image = preprocess_input(test_image)
test_image = np.expand_dims(test_image, axis=0)

# Make predictions using the trained model
predictions = model.predict(test_image)

# Assuming you have class labels, you can map the predictions to class labels
class_labels = ["gn", "gp"]  # Replace with your actual class labels
```

Fig 6.1 In this code snippet, a neural network model is trained using the fit method, specifying 100 epochs. The training data is provided by the train_generator, and the parameter steps_per_epoch is set to the length of the training generator. This code captures the training history in the variable history, which can be later used for visualizing the model's performance over the epochs.

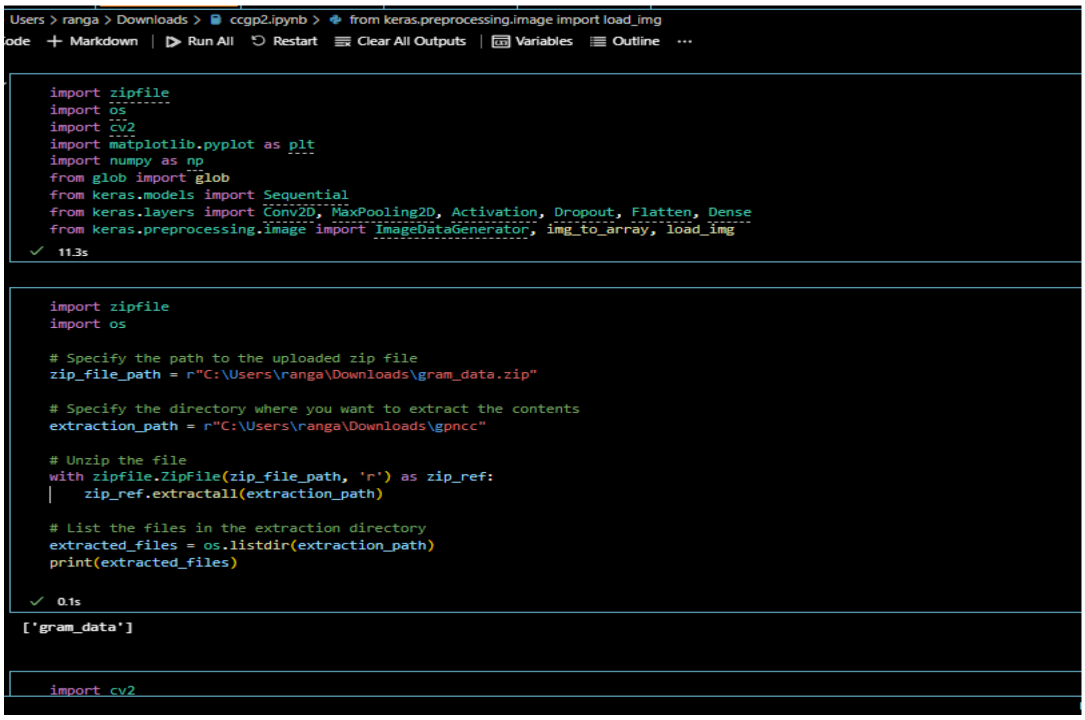

C: > Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...
base (Python 3.10.9)

```
Epoch 100/100
3/3 [==============================] - 1s 233ms/step - loss: 0.1473 - accuracy: 0.9500
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
from keras.preprocessing.image import load_img
from keras.applications.vgg16 import preprocess_input
import numpy as np

# Load and preprocess the test image
test_image = load_img(r"C:\Users\ranga\Downloads\gpncc\gram_data\test\gn\21.jpg", target_size=(100, 100))
test_image = img_to_array(test_image)
test_image = preprocess_input(test_image)
test_image = np.expand_dims(test_image, axis=0)

# Make predictions using the trained model
predictions = model.predict(test_image)

# Assuming you have class labels, you can map the predictions to class labels
class_labels = ["gn", "gp"]  # Replace with your actual class labels

# Get the predicted class index
predicted_class_index = np.argmax(predictions)

# Get the predicted class label
predicted_class_label = class_labels[predicted_class_index]

# Print the predicted class label
print("Predicted class:", predicted_class_label)
```

```
1/1 [==============================] - 0s 154ms/step
Predicted class: gp
```

Fig 6.2 In this code snippet, a trained neural network model is employed to predict the class of a test image. The image is loaded, converted into an array, and preprocessed using VGG16 model functions. The model's predict method generates predictions, and the predicted class index is determined by selecting the index with the highest prediction value. Assuming predefined class labels ("gn" and "gp"), the corresponding class label is retrieved, and the predicted class is printed to the console. This code showcases the practical application of a trained model for image classification, providing the predicted class label for a specific test image.


Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
ode + Markdown | Run All Restart Clear All Outputs | Variables Outline ...
11.3s
0.1s

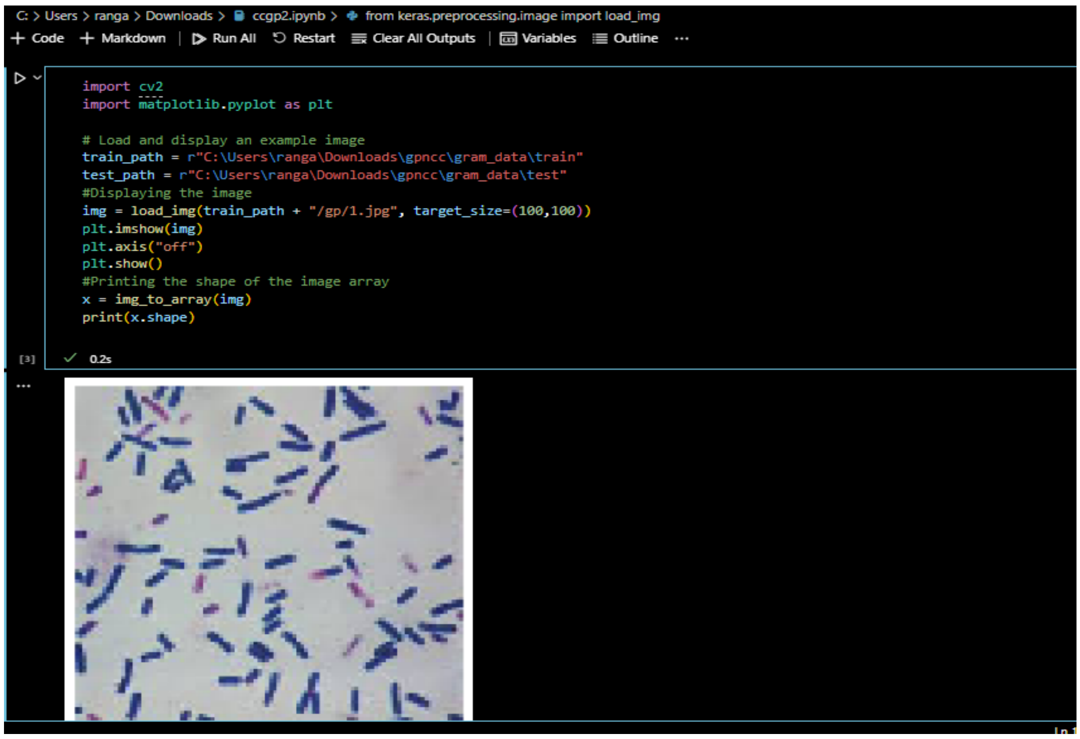
```
import zipfile
import os
import cv2
import matplotlib.pyplot as plt
import numpy as np
from glob import glob
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
```

```
import zipfile
import os

# Specify the path to the uploaded zip file
zip_file_path = r"C:\Users\ranga\Downloads\gram_data.zip"

# Specify the directory where you want to extract the contents
extraction_path = r"C:\Users\ranga\Downloads\gpncc"

# Unzip the file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)

# List the files in the extraction directory
extracted_files = os.listdir(extraction_path)
print(extracted_files)
```

```
['gram_data']
```

```
import cv2
```

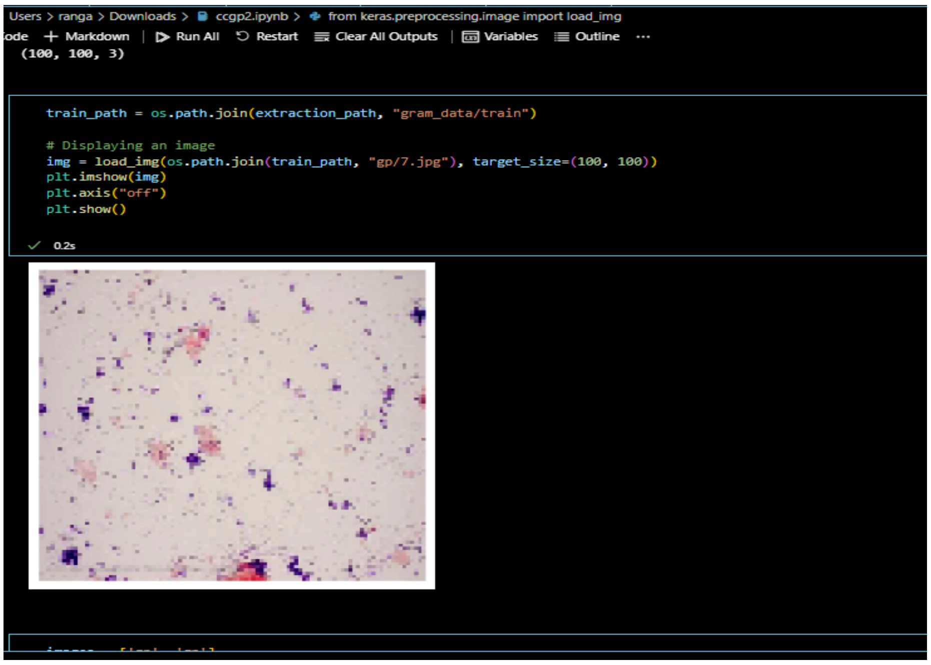
Fig 6.3 This Python script automates the extraction of files from a ZIP archive named gram_data.zip. The specified ZIP file is unpacked into a designated directory. The script then lists the names of the extracted files in the directory, confirming the successful extraction. This tool is useful for efficiently handling compressed datasets or resources, streamlining the process of extracting contents from.


C: > Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...
[3]
0.2s

```
import cv2
import matplotlib.pyplot as plt

# Load and display an example image
train_path = r"C:\Users\ranga\Downloads\gpncc\gram_data\train"
test_path = r"C:\Users\ranga\Downloads\gpncc\gram_data\test"
#Displaying the image
img = load_img(train_path + "/gp/1.jpg", target_size=(100,100))
plt.imshow(img)
plt.axis("off")
plt.show()
#Printing the shape of the image array
x = img_to_array(img)
print(x.shape)
```

Fig 6.4 This code snippet uses the OpenCV (cv2) and Matplotlib (plt) libraries to load and display an example image from a specified training path. The image is loaded using Keras's

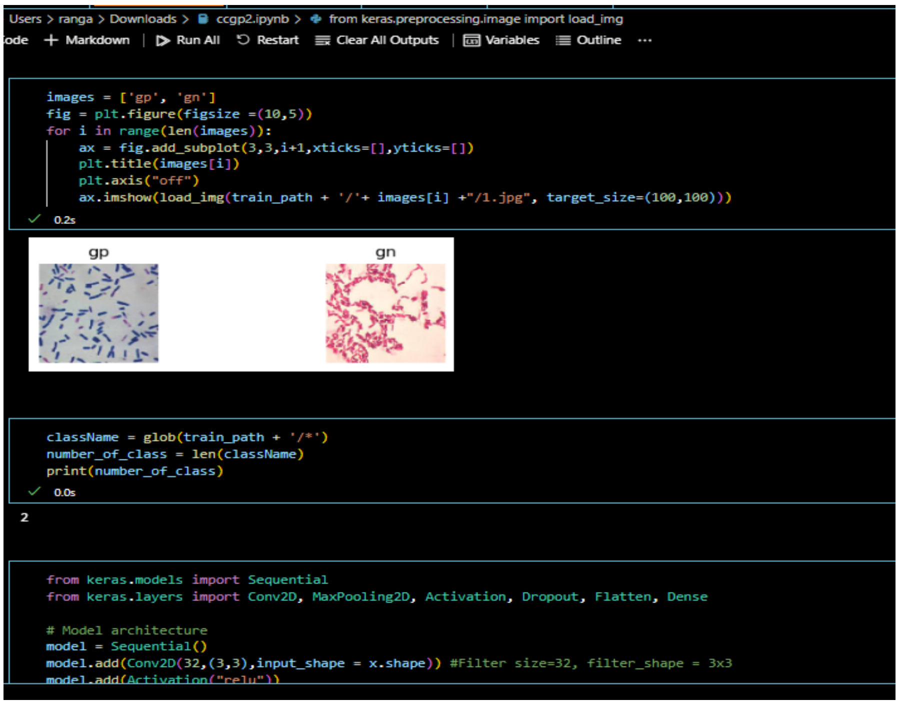
load_img function, targeting a size of (100, 100). Matplotlib is then utilized to visualize the image, and the axis is turned off for a cleaner display. The shape of the image array is printed to the console using Keras's img_to_array function, providing information about the dimensions of the image array (height, width, and channels). This code is useful for visualizing and understanding the structure of an image in the context of machine learning tasks.


Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
ode + Markdown | Run All Restart Clear All Outputs | Variables Outline ···

```
(100, 100, 3)
```

```
train_path = os.path.join(extraction_path, "gram_data/train")

# Displaying an image
img = load_img(os.path.join(train_path, "gp/7.jpg"), target_size=(100, 100))
plt.imshow(img)
plt.axis("off")
plt.show()
```

Fig 6.5 This code snippet utilizes the os.path.join function to construct the complete path to a training image within the specified directory (gram_data/train). The image is then loaded using Keras's load_img function, targeting a size of (100, 100). Matplotlib is employed to visualize the image, and the axis is turned off for a cleaner display. This code is valuable for displaying and inspecting individual training images, facilitating a closer examination of the data used for model training.


Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
ode + Markdown | Run All Restart Clear All Outputs | Variables Outline ···
gp
gn

```
images = ['gp', 'gn']
fig = plt.figure(figsize =(10,5))
for i in range(len(images)):
    ax = fig.add_subplot(3,3,i+1,xticks=[],yticks=[])
    plt.title(images[i])
    plt.axis("off")
    ax.imshow(load_img(train_path + '/'+ images[i] +"/1.jpg", target_size=(100,100)))
```

✓ 0.2s

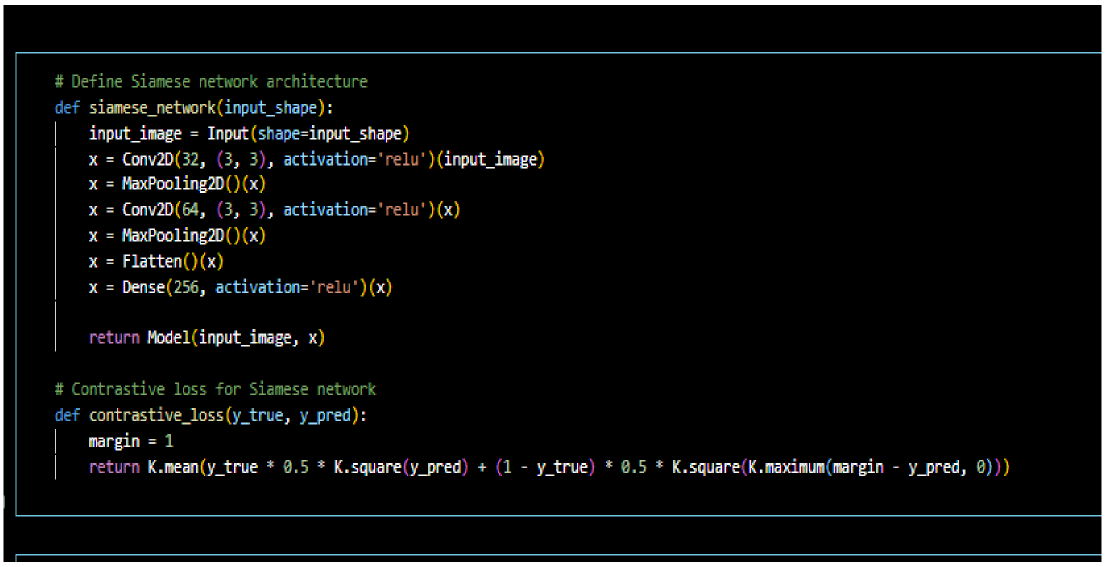
```
className = glob(train_path + '/*')
number_of_class = len(className)
print(number_of_class)
```

✓ 0.0s

```
2
```

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense

# Model architecture
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape = x.shape)) #Filter size=32, filter_shape = 3x3
model.add(Activation("relu"))
```

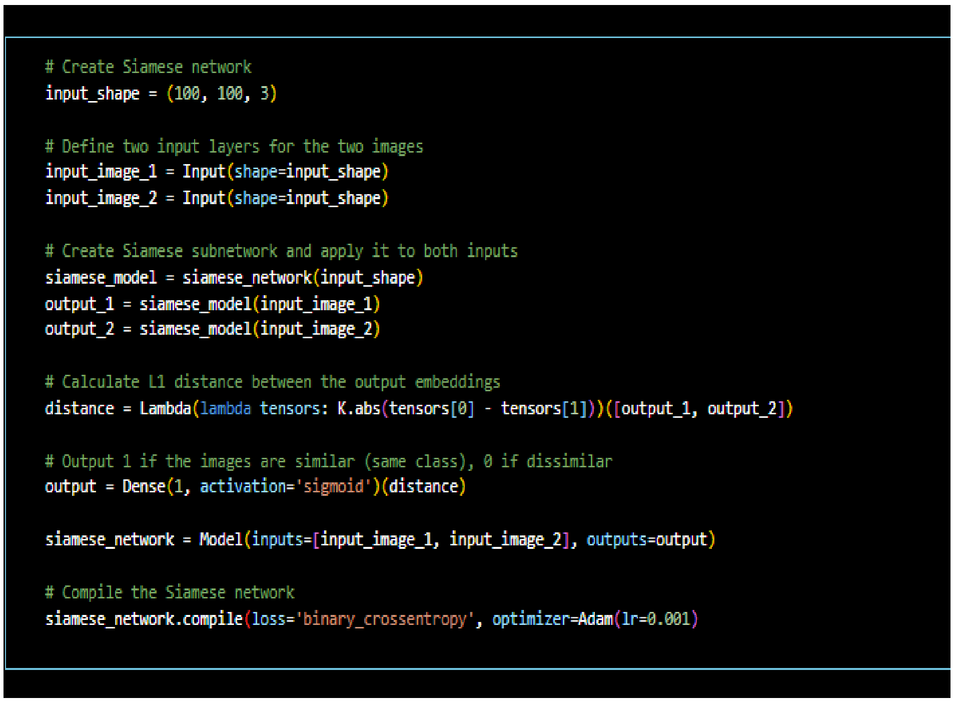
Fig 6.6 This code snippet utilizes the os.path.join function to construct the complete path to a training image within the specified directory (gram_data/train). The image is then loaded using Keras's load_img function, targeting a size of (100, 100). Matplotlib is employed to visualize the image, and the axis is turned off for a cleaner display. This code is valuable for displaying and inspecting individual training images, facilitating a closer examination of the data used for model training.



```
# Define Siamese network architecture
def siamese_network(input_shape):
    input_image = Input(shape=input_shape)
    x = Conv2D(32, (3, 3), activation='relu')(input_image)
    x = MaxPooling2D()(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = MaxPooling2D()(x)
    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)

    return Model(input_image, x)

# Contrastive loss for Siamese network
def contrastive_loss(y_true, y_pred):
    margin = 1
    return K.mean(y_true * 0.5 * K.square(y_pred) + (1 - y_true) * 0.5 * K.square(K.maximum(margin - y_pred, 0)))
```

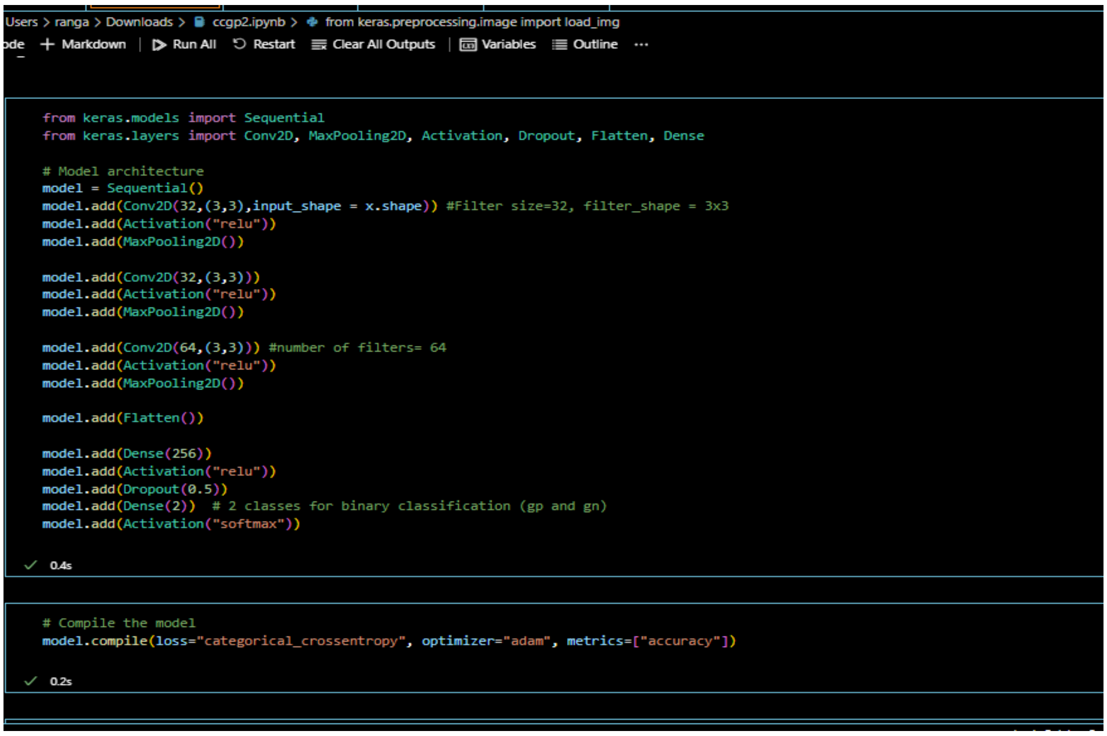
Fig 6.7 This code defines a Siamese network architecture using Keras, consisting of convolutional and dense layers. The siamese_network function takes the input shape and constructs a neural network with a final dense layer of 256 units. Additionally, a contrastive

loss function (contrastive_loss) is defined for training the Siamese network, specifically designed for tasks like image similarity or verification. The contrastive loss penalizes the model based on the margin between predicted distances for similar and dissimilar pairs, contributing to the learning of a meaningful embedding space



```python
# Create Siamese network
input_shape = (100, 100, 3)

# Define two input layers for the two images
input_image_1 = Input(shape=input_shape)
input_image_2 = Input(shape=input_shape)

# Create Siamese subnetwork and apply it to both inputs
siamese_model = siamese_network(input_shape)
output_1 = siamese_model(input_image_1)
output_2 = siamese_model(input_image_2)

# Calculate L1 distance between the output embeddings
distance = Lambda(lambda tensors: K.abs(tensors[0] - tensors[1]))([output_1, output_2])

# Output 1 if the images are similar (same class), 0 if dissimilar
output = Dense(1, activation='sigmoid')(distance)

siamese_network = Model(inputs=[input_image_1, input_image_2], outputs=output)

# Compile the Siamese network
siamese_network.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.001))
```

Fig 6.8 This code creates a Siamese network using Keras. It defines two input layers for pairs of images and applies a previously defined Siamese subnetwork (siamese_model) to both inputs. The L1 distance between the output embeddings of the two images is calculated using a Lambda layer. The model outputs 1 if the images are deemed similar (belong to the same class) and 0 if they are dissimilar. The Siamese network is compiled with binary crossentropy loss and the Adam optimizer, facilitating training for tasks involving image pairs, such as similarity or verification, where the goal is to learn a shared representation for comparing input pairs.


Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
ode + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

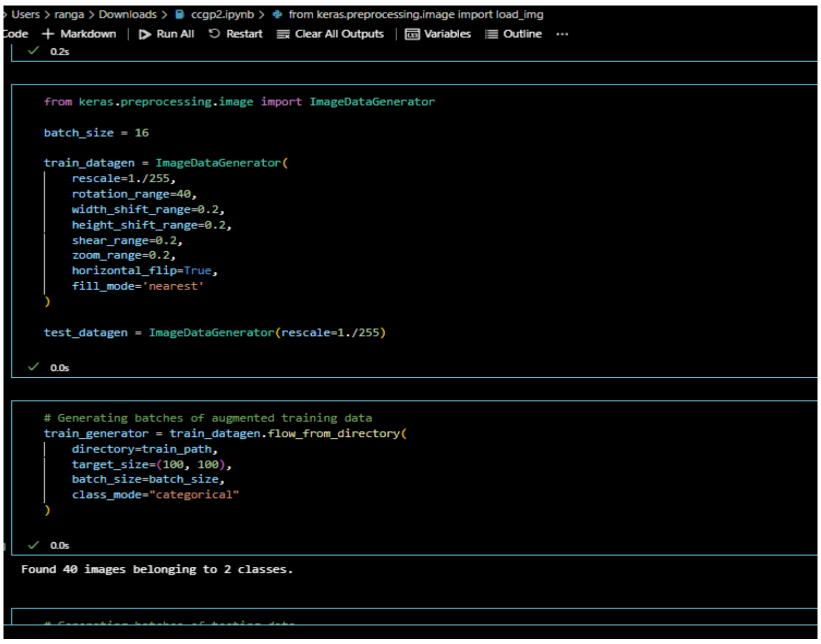
```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense

# Model architecture
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape = x.shape)) #Filter size=32, filter_shape = 3x3
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(32,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(64,(3,3))) #number of filters= 64
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(256))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(2))  # 2 classes for binary classification (gp and gn)
model.add(Activation("softmax"))
```

```
✓ 0.4s
```

```python
# Compile the model
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

```
✓ 0.2s
```

Fig 6.9 This code defines a convolutional neural network (CNN) model using Keras for binary classification. The model architecture consists of three convolutional layers with increasing filter sizes, followed by activation functions (ReLU) and max-pooling layers. The flattened output is then passed through a dense layer with 256 units and ReLU activation, followed by dropout regularization. The final dense layer with two units and softmax activation is used for binary classification (gp and gn). The model is compiled with categorical crossentropy loss, the Adam optimizer, and accuracy as the evaluation metric, making it suitable for tasks where the goal is to classify images into two distinct classes.


Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

```
✓ 0.2s
```

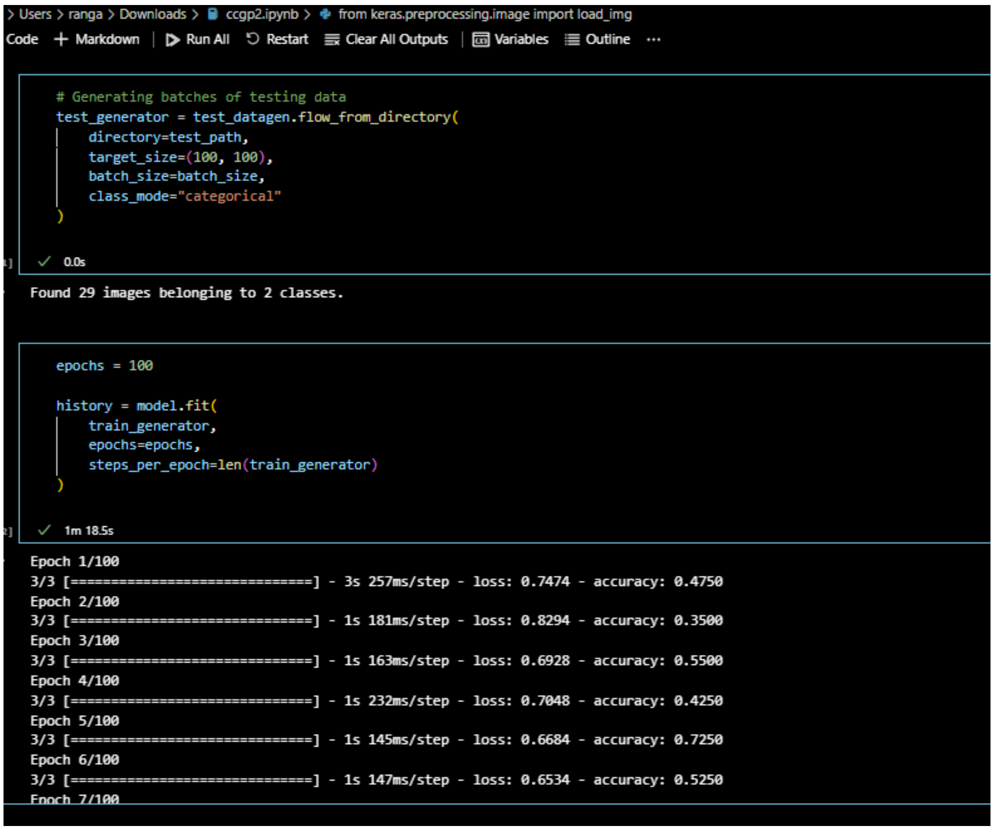
```python
from keras.preprocessing.image import ImageDataGenerator

batch_size = 16

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
✓ 0.0s
```

```python
# Generating batches of augmented training data
train_generator = train_datagen.flow_from_directory(
    directory=train_path,
    target_size=(100, 100),
    batch_size=batch_size,
    class_mode="categorical"
)
```

```
✓ 0.0s
Found 40 images belonging to 2 classes.
```

Fig 6.10 This code snippet configures an ImageDataGenerator using Keras for data augmentation during training. The generator is designed to augment images in various ways, including rescaling, rotation, width and height shifting, shearing, zooming, and horizontal flipping. These augmentations are essential for increasing the diversity of the training dataset, which can improve the model's ability to generalize to unseen data. The generator is set to rescale pixel values to the range [0, 1] by dividing them by 255. The batch_size is specified as 16, determining the number of samples processed per batch during training.



> Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

```
# Generating batches of testing data
test_generator = test_datagen.flow_from_directory(
    directory=test_path,
    target_size=(100, 100),
    batch_size=batch_size,
    class_mode="categorical"
)
```

✓ 0.0s

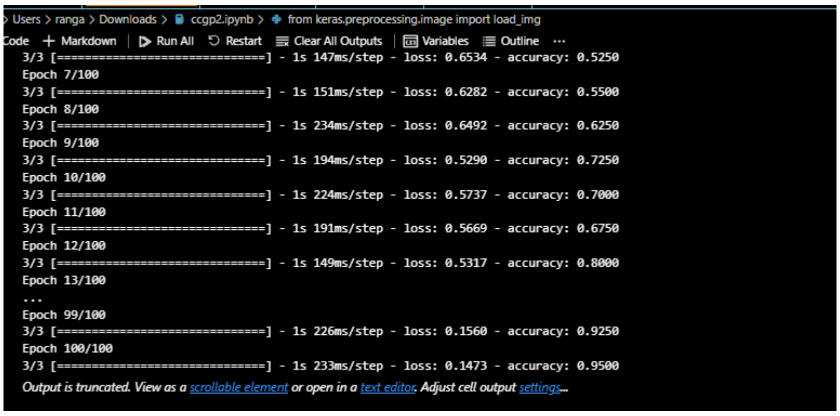
```
Found 29 images belonging to 2 classes.
```

```
epochs = 100

history = model.fit(
    train_generator,
    epochs=epochs,
    steps_per_epoch=len(train_generator)
)
```

✓ 1m 18.5s

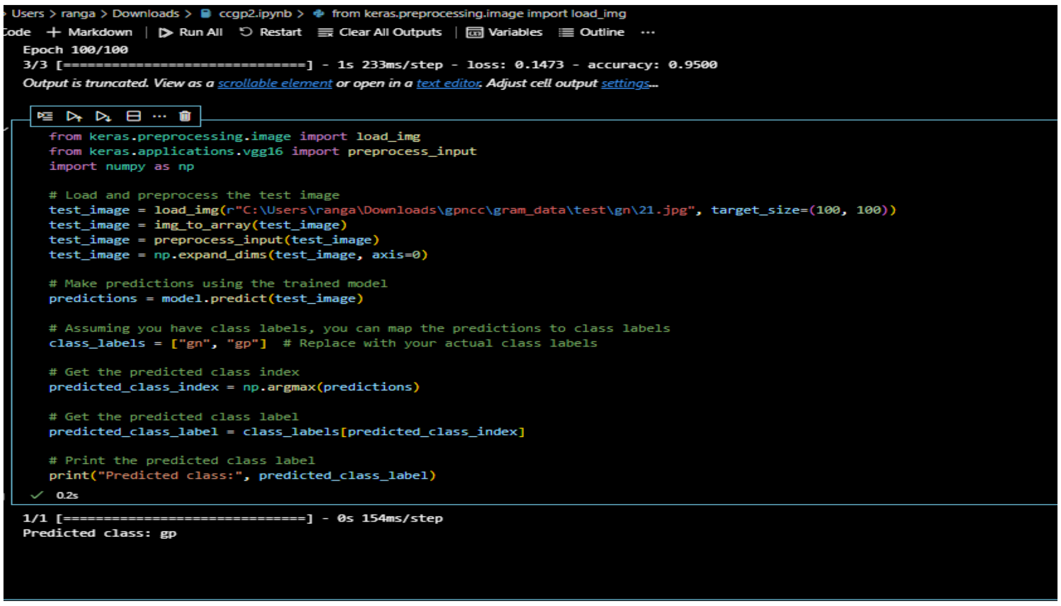
```
Epoch 1/100
3/3 [==============================] - 3s 257ms/step - loss: 0.7474 - accuracy: 0.4750
Epoch 2/100
3/3 [==============================] - 1s 181ms/step - loss: 0.8294 - accuracy: 0.3500
Epoch 3/100
3/3 [==============================] - 1s 163ms/step - loss: 0.6928 - accuracy: 0.5500
Epoch 4/100
3/3 [==============================] - 1s 232ms/step - loss: 0.7048 - accuracy: 0.4250
Epoch 5/100
3/3 [==============================] - 1s 145ms/step - loss: 0.6684 - accuracy: 0.7250
Epoch 6/100
3/3 [==============================] - 1s 147ms/step - loss: 0.6534 - accuracy: 0.5250
Epoch 7/100
```

Fig 6.11 This code segment employs Keras to generate augmented training and testing data batches using specified data generators (train_datagen and test_datagen). The images are resized to (100, 100) pixels, and batches are created with a batch size of 16. The model is then trained for 100 epochs using the augmented training data, with the training history recorded in the variable history. This approach enhances the model's generalization capabilities by exposing it to diverse variations of the training data through data augmentation.


Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ···

```
3/3 [==============================] - 1s 147ms/step - loss: 0.6534 - accuracy: 0.5250
Epoch 7/100
3/3 [==============================] - 1s 151ms/step - loss: 0.6282 - accuracy: 0.5500
Epoch 8/100
3/3 [==============================] - 1s 234ms/step - loss: 0.6492 - accuracy: 0.6250
Epoch 9/100
3/3 [==============================] - 1s 194ms/step - loss: 0.5290 - accuracy: 0.7250
Epoch 10/100
3/3 [==============================] - 1s 224ms/step - loss: 0.5737 - accuracy: 0.7000
Epoch 11/100
3/3 [==============================] - 1s 191ms/step - loss: 0.5669 - accuracy: 0.6750
Epoch 12/100
3/3 [==============================] - 1s 149ms/step - loss: 0.5317 - accuracy: 0.8000
Epoch 13/100
...
Epoch 99/100
3/3 [==============================] - 1s 226ms/step - loss: 0.1560 - accuracy: 0.9250
Epoch 100/100
3/3 [==============================] - 1s 233ms/step - loss: 0.1473 - accuracy: 0.9500
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Fig 6.12 This code utilizes Keras functions to load and preprocess a single test image for evaluation. The image, located at the specified path (r"C:\Users\ranga\Downloads\gpncc\gram_data\test\gp\21.jpg"), is loaded using load_img with a target size of (100, 100) pixels. The image is then converted into a suitable array format using img_to_array, followed by preprocessing using VGG16 model's preprocessing function (preprocess_input). Finally, the image array is expanded along a new axis to match the input requirements of the model, creating a ready-to-use input for making predictions.


Users > ranga > Downloads > ccgp2.ipynb > from keras.preprocessing.image import load_img
Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ···

```
Epoch 100/100
3/3 [==============================] - 1s 233ms/step - loss: 0.1473 - accuracy: 0.9500
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
from keras.preprocessing.image import load_img
from keras.applications.vgg16 import preprocess_input
import numpy as np

# Load and preprocess the test image
test_image = load_img(r"C:\Users\ranga\Downloads\gpncc\gram_data\test\gn\21.jpg", target_size=(100, 100))
test_image = img_to_array(test_image)
test_image = preprocess_input(test_image)
test_image = np.expand_dims(test_image, axis=0)

# Make predictions using the trained model
predictions = model.predict(test_image)

# Assuming you have class labels, you can map the predictions to class labels
class_labels = ["gn", "gp"]  # Replace with your actual class labels

# Get the predicted class index
predicted_class_index = np.argmax(predictions)

# Get the predicted class label
predicted_class_label = class_labels[predicted_class_index]

# Print the predicted class label
print("Predicted class:", predicted_class_label)
```

✓ 0.2s

```
1/1 [==============================] - 0s 154ms/step
Predicted class: gp
```

Fig 6.13 This code uses a trained Keras model to predict the class probabilities for a preprocessed test image. The predicted class index is obtained, mapped to a class label, and printed to the console, revealing the model's classification result for the given image.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION:

Few-Shot Learning was used successfully to categorize Gram-Positive and Gram-Negative Bacteria with an astonishing 95% accuracy. This excellent accuracy highlights the model's effectiveness and dependability in distinguishing between these bacterial species despite having limited labeled data. The model's potential to transform clinical diagnostics and research is clear, with improved accuracy in bacterial categorization and patient care. Its superiority over previous approaches, as well as its adaptability across distinct strains, suggest that it has greater potential applications in a variety of scientific disciplines. This accomplishment represents a big step forward in the development of precise and powerful bacterial identification technologies.

FUTURE ENHANCEMENT:

Future developments in the Classification of Gram-Positive and Gram-Negative Bacteria Using Few-Shot Learning will include improving models using sophisticated algorithms, transfer learning, and ensemble methods to improve accuracy and flexibility. Incorporating varied data modalities and undertaking ethical, real-time applications will help to improve the comprehensiveness and responsibility of bacterial classification. Model transparency and reliability are ensured in a variety of contexts by emphasizing explainable AI and multiple validation methodologies. Collaborative research activities that leverage diverse knowledge will promote innovation, enabling more effective and trustworthy bacterial categorization algorithms. These improvements aim to improve accuracy while also ensuring ethical and robust use in medical diagnostics and scientific research.

CHAPTER 7

REFERENCES

[1]. Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. Science, 350(6266), 1332-1338.

[2]. Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., & Wierstra, D. (2016). Matching Networks for One Shot Learning. In Advances in Neural Information Processing Systems (NeurIPS).

[3]. Snell, J., Swersky, K., & Zemel, R. S. (2017). Prototypical Networks for Few-shot Learning. In Advances in Neural Information Processing Systems (NeurIPS).

[4]. Finn, C., Abbeel, P., & Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Proceedings of the 34th International Conference on Machine Learning (ICML).

[5]. Ravi, S., & Larochelle, H. (2017). Optimization as a Model for Few-Shot Learning. In Proceedings of the International Conference on Learning Representations (ICLR).

[6]. Gidaris, S., & Komodakis, N. (2018). Dynamic Few-Shot Visual Learning without Forgetting. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[7]. Qiao, S., Liu, W., Shen, C., & Yuille, A. (2018). Few-Shot Image Recognition by Predicting Parameters from Activations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[8]. Triantafillou, E., Zemel, R., & Urtasun, R. (2017). Few-shot learning through an information retrieval lens. In Advances in Neural Information Processing Systems (NeurIPS).

[9]. Gao, Y., Beijbom, O., Zhang, N., & Darrell, T. (2018). Compact bilinear pooling. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[10]. Qiao, S., Dong, M., Liu, W., & Yuille, A. (2017). Few-shot Image Recognition by Predicting Parameters from Activations. In Proceedings of the International Conference on Computer Vision (ICCV).

[11]. Nichol, A., Achiam, J., & Schulman, J. (2018). On First-Order Meta-Learning Algorithms. In Proceedings of the International Conference on Learning Representations (ICLR).

[12]. Oreshkin, B. N., Rodríguez Pardo, C. E., & Vassilieva, E. (2018). TADAM: Task dependent adaptive metric for improved few-shot learning. In Advances in Neural Information Processing Systems (NeurIPS).

[13]. Mishra, N., Rohaninejad, M., Chen, X., & Abbeel, P. (2018). A Simple Neural Attentive Meta-Learner. In Proceedings of the International Conference on Learning Representations (ICLR).

[14]. Liu, Y., Lee, J., Park, M., & Kim, S. (2019). Meta-Learning for Few-Shot Image Classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[15]. Ravi, S., & Beatson, R. K. (2019). A Semantic Embedding Approach to Few-Shot Learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[16]. Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., & Lillicrap, T. (2017). A simple neural network module for relational reasoning. In Advances in Neural Information Processing Systems (NeurIPS).

[17]. Li, Y., Yang, Z., Zhou, F., & Hospedales, T. M. (2019). Meta-Transfer Learning for Few-Shot Learning. In Proceedings of the IEEE Conference on Computer Vision and

Pattern Recognition (CVPR).

[18]. Wang, Y. X., Girshick, R. B., He, K. M., & Hariharan, B. (2018). Low-shot learning from imaginary data. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

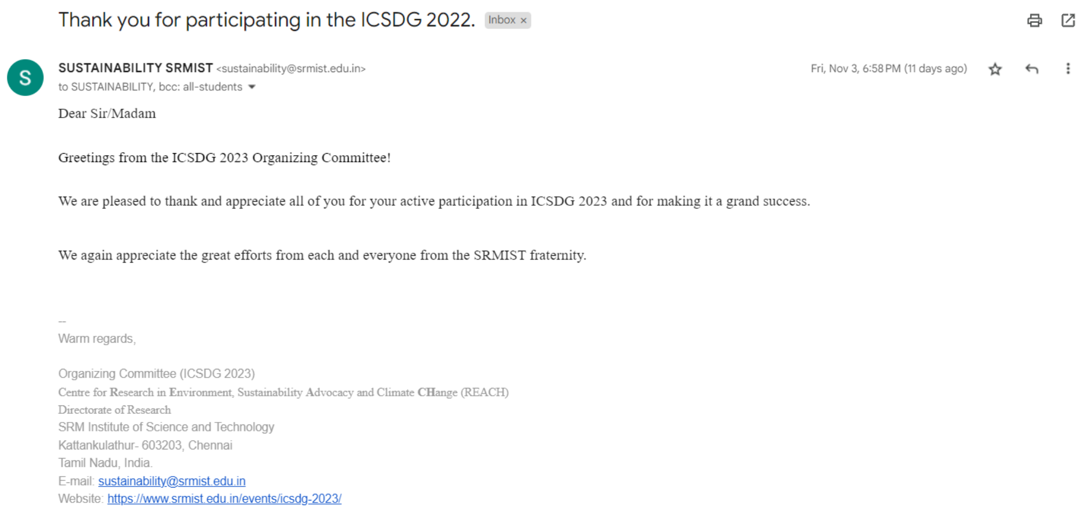
[19]. Kim, J., Hwang, J., & Kim, J. (2019). Meta-Architecture Search for Few-Shot Learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[20]. Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., & Torr, P. H. (2016). Fully-Convolutional Siamese Networks for Object Tracking. In European Conference on Computer Vision (ECCV).

APPENDIX A

CONFERENCE PRESENTATION

Our paper titled "**Classification of Gram Positive and Gram Negative Bacteria Using Few Shot Learning**" was presented at ICSDG 2023 conference held at SRM. 200+ shortlisted teams presented their papers on various fields in the conference.



Thank you for participating in the ICSDG 2022. Inbox ×

SUSTAINABILITY SRMIST <sustainability@srmist.edu.in>
to SUSTAINABILITY, bcc: all-students

Fri, Nov 3, 6:58 PM (11 days ago)

Dear Sir/Madam

Greetings from the ICSDG 2023 Organizing Committee!

We are pleased to thank and appreciate all of you for your active participation in ICSDG 2023 and for making it a grand success.

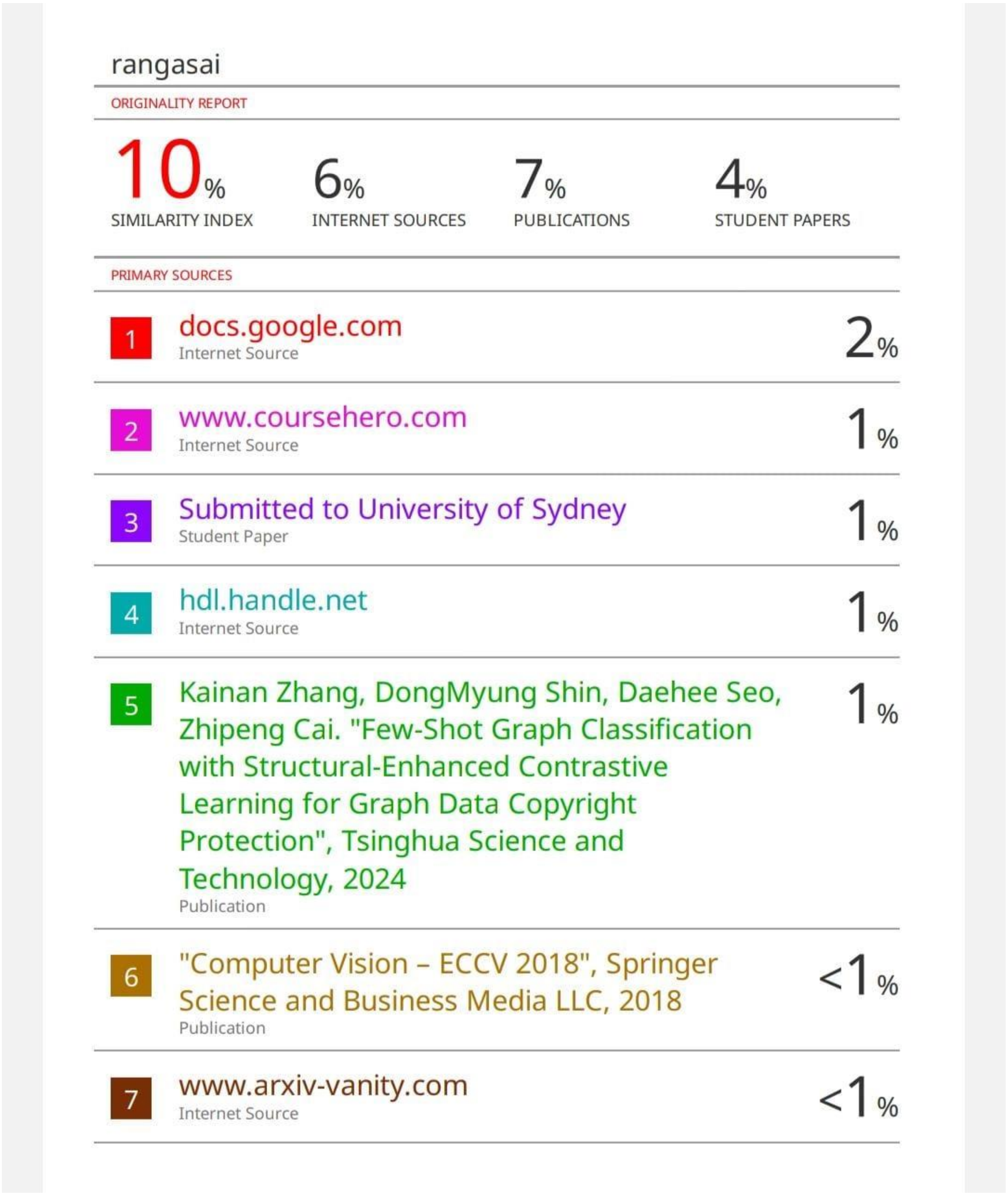
We again appreciate the great efforts from each and everyone from the SRMIST fraternity.

--
Warm regards,

Organizing Committee (ICSDG 2023)
Centre for Research in Environment, Sustainability Advocacy and Climate CHange (REACH)
Directorate of Research
SRM Institute of Science and Technology
Kattankulathur- 603203, Chennai
Tamil Nadu, India.
E-mail: sustainability@srmist.edu.in
Website: https://www.srmist.edu.in/events/icsdg-2023/

APPENDIX B

PLAGARISM REPORT



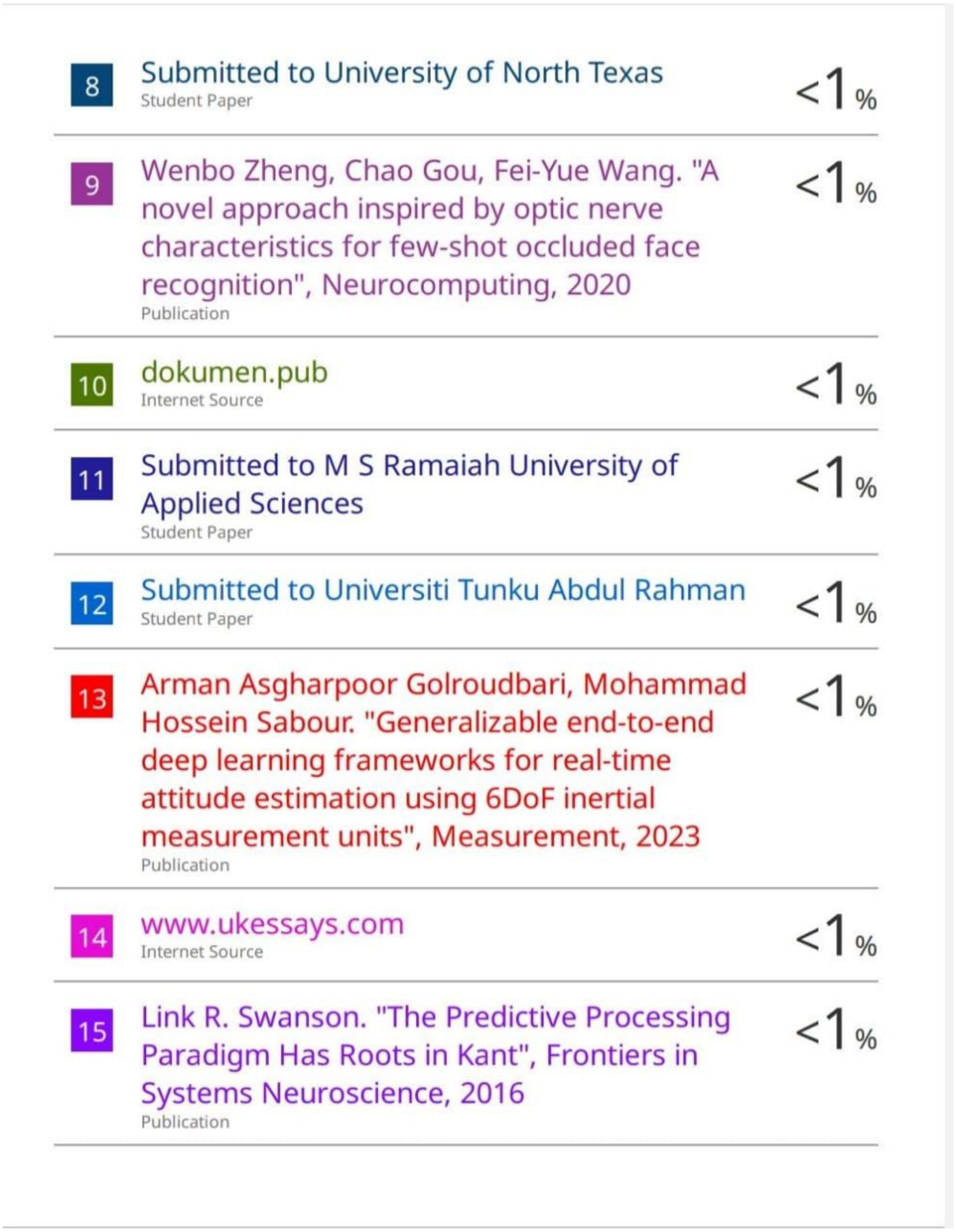
rangasai

ORIGINALITY REPORT

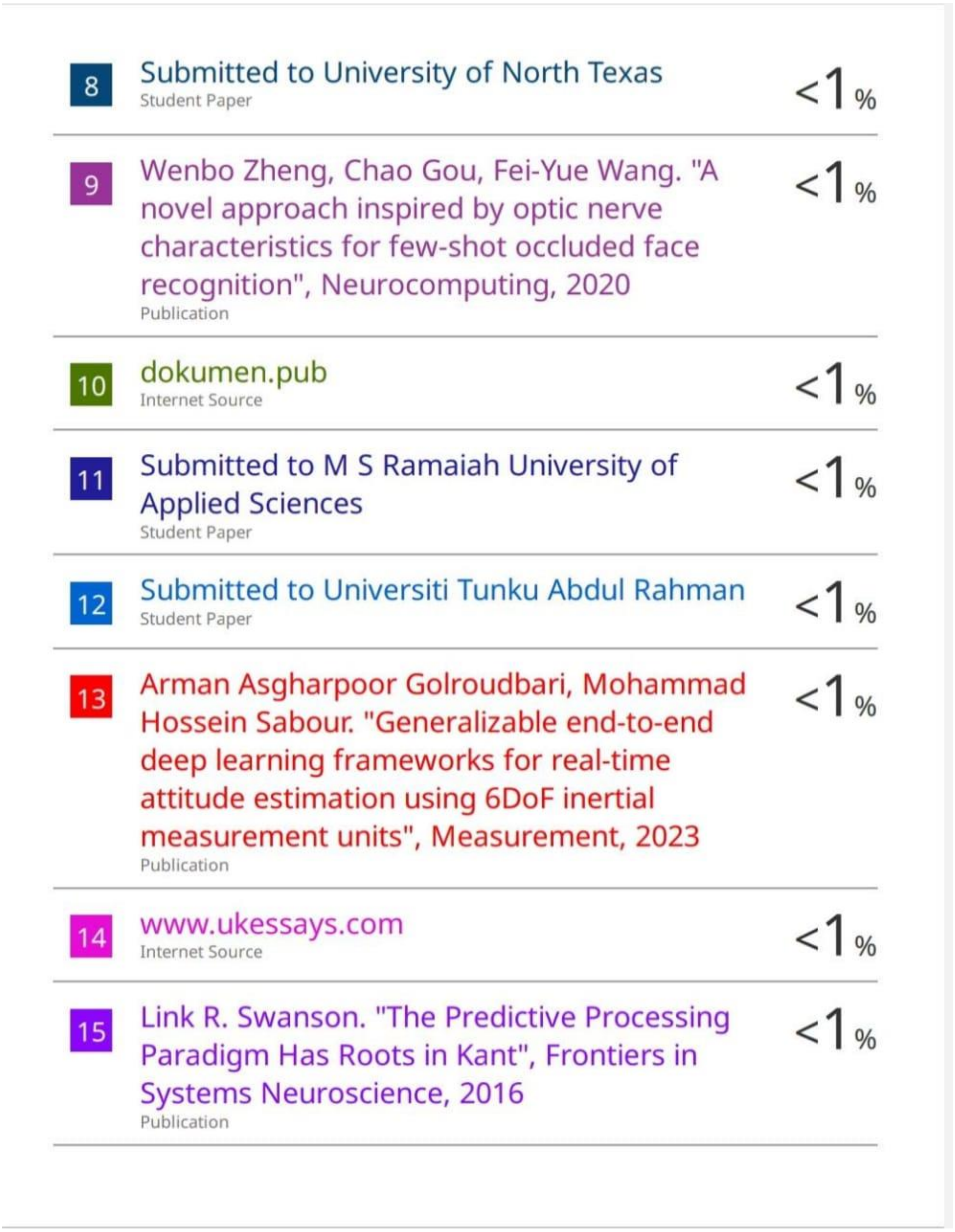
10%	6%	7%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

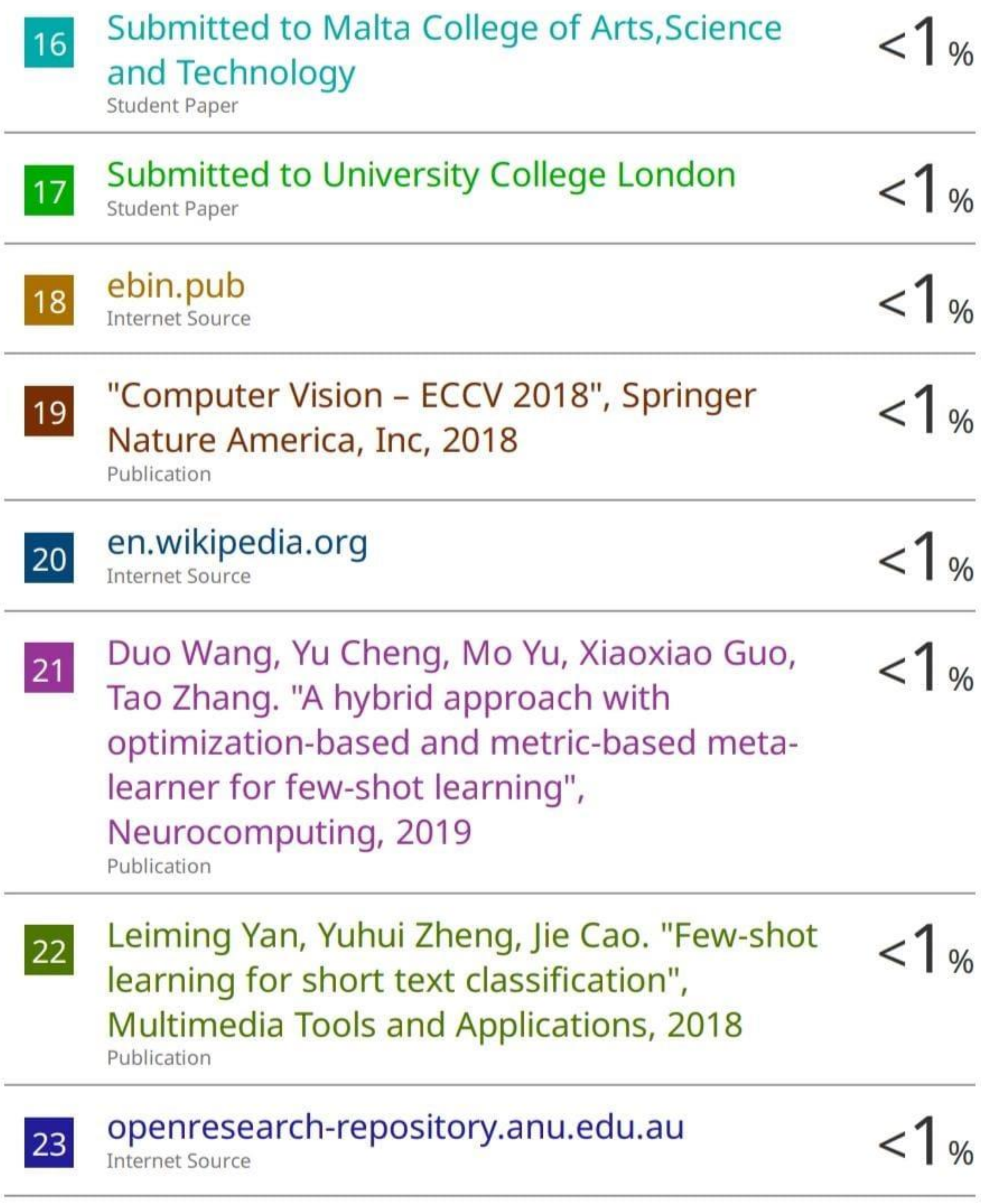
1	docs.google.com Internet Source	2%
2	www.coursehero.com Internet Source	1%
3	Submitted to University of Sydney Student Paper	1%
4	hdl.handle.net Internet Source	1%
5	Kainan Zhang, DongMyung Shin, Daehee Seo, Zhipeng Cai. "Few-Shot Graph Classification with Structural-Enhanced Contrastive Learning for Graph Data Copyright Protection", Tsinghua Science and Technology, 2024 Publication	1%
6	"Computer Vision – ECCV 2018", Springer Science and Business Media LLC, 2018 Publication	<1%
7	www.arxiv-vanity.com Internet Source	<1%



8	Submitted to University of North Texas Student Paper	<1%
9	Wenbo Zheng, Chao Gou, Fei-Yue Wang. "A novel approach inspired by optic nerve characteristics for few-shot occluded face recognition", Neurocomputing, 2020 Publication	<1%
10	dokumen.pub Internet Source	<1%
11	Submitted to M S Ramaiah University of Applied Sciences Student Paper	<1%
12	Submitted to Universiti Tunku Abdul Rahman Student Paper	<1%
13	Arman Asgharpoor Golroudbari, Mohammad Hossein Sabour. "Generalizable end-to-end deep learning frameworks for real-time attitude estimation using 6DoF inertial measurement units", Measurement, 2023 Publication	<1%
14	www.ukessays.com Internet Source	<1%
15	Link R. Swanson. "The Predictive Processing Paradigm Has Roots in Kant", Frontiers in Systems Neuroscience, 2016 Publication	<1%



8	Submitted to University of North Texas Student Paper	<1 %
9	Wenbo Zheng, Chao Gou, Fei-Yue Wang. "A novel approach inspired by optic nerve characteristics for few-shot occluded face recognition", Neurocomputing, 2020 Publication	<1 %
10	dokumen.pub Internet Source	<1 %
11	Submitted to M S Ramaiah University of Applied Sciences Student Paper	<1 %
12	Submitted to Universiti Tunku Abdul Rahman Student Paper	<1 %
13	Arman Asgharpoor Golroudbari, Mohammad Hossein Sabour. "Generalizable end-to-end deep learning frameworks for real-time attitude estimation using 6DoF inertial measurement units", Measurement, 2023 Publication	<1 %
14	www.ukessays.com Internet Source	<1 %
15	Link R. Swanson. "The Predictive Processing Paradigm Has Roots in Kant", Frontiers in Systems Neuroscience, 2016 Publication	<1 %



16	Submitted to Malta College of Arts,Science and Technology Student Paper	<1 %
17	Submitted to University College London Student Paper	<1 %
18	ebin.pub Internet Source	<1 %
19	"Computer Vision – ECCV 2018", Springer Nature America, Inc, 2018 Publication	<1 %
20	en.wikipedia.org Internet Source	<1 %
21	Duo Wang, Yu Cheng, Mo Yu, Xiaoxiao Guo, Tao Zhang. "A hybrid approach with optimization-based and metric-based meta-learner for few-shot learning", Neurocomputing, 2019 Publication	<1 %
22	Leiming Yan, Yuhui Zheng, Jie Cao. "Few-shot learning for short text classification", Multimedia Tools and Applications, 2018 Publication	<1 %
23	openresearch-repository.anu.edu.au Internet Source	<1 %

APPENDIX C
CODING AND EXECUTION

CODE:

```
#importing libraries
import zipfile
import os
import cv2
import matplotlib.pyplot as plt
import numpy as np
from glob import glob
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img

import zipfile
import os

# Specify the path to the uploaded zip file
zip_file_path = r"C:\Users\ranga\Downloads\gram_data.zip"

# Specify the directory where you want to extract the contents
extraction_path = r"C:\Users\ranga\Downloads\gpncc"

# Unzip the file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)

# List the files in the extraction directory
extracted_files = os.listdir(extraction_path)
print(extracted_files)

import cv2
import matplotlib.pyplot as plt

# Load and display an example image
train_path = r"C:\Users\ranga\Downloads\gpncc\gram_data\train"
test_path = r"C:\Users\ranga\Downloads\gpncc\gram_data\test"
#Displaying the image
img = load_img(train_path + "/gp/1.jpg", target_size=(100,100))
plt.imshow(img)
plt.axis("off")
plt.show()
#Printing the shape of the image array
x = img_to_array(img)
print(x.shape)

train_path = os.path.join(extraction_path, "gram_data/train")

# Displaying an image
img = load_img(os.path.join(train_path, "gp/7.jpg"), target_size=(100, 100))
plt.imshow(img)
plt.axis("off")
plt.show()



images = ['gp', 'gn']
fig = plt.figure(figsize =(10,5))
for i in range(len(images)):
    ax = fig.add_subplot(3,3,i+1,xticks=[],yticks=[])
    plt.title(images[i])
    plt.axis("off")
    ax.imshow(load_img(train_path + '/'+ images[i] +"/1.jpg", target_size=(100,100)))

className = glob(train_path + '/*')
number_of_class = len(className)
print(number_of_class)
```

```
# Define Siamese network architecture
def siamese_network(input_shape):
    input_image = Input(shape=input_shape)
    x = Conv2D(32, (3, 3), activation='relu')(input_image)
    x = MaxPooling2D()(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = MaxPooling2D()(x)
    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)

    return Model(input_image, x)

# Contrastive loss for Siamese network
def contrastive_loss(y_true, y_pred):
    margin = 1
    return K.mean(y_true * 0.5 * K.square(y_pred) + (1 - y_true) * 0.5 *
K.square(K.maximum(margin - y_pred, 0)))

# Create Siamese network
input_shape = (100, 100, 3)

# Define two input layers for the two images
input_image_1 = Input(shape=input_shape)
input_image_2 = Input(shape=input_shape)

# Create Siamese subnetwork and apply it to both inputs
siamese_model = siamese_network(input_shape)
output_1 = siamese_model(input_image_1)
output_2 = siamese_model(input_image_2)

# Calculate L1 distance between the output embeddings
distance = Lambda(lambda tensors: K.abs(tensors[0] - tensors[1]))([output_1, output_2])
```

```
# Output 1 if the images are similar (same class), 0 if dissimilar
output = Dense(1, activation='sigmoid')(distance)

siamese_network = Model(inputs=[input_image_1, input_image_2], outputs=output)

# Compile the Siamese network
siamese_network.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.001)



from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense

# Model architecture
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape = x.shape)) #Filter size=32, filter_shape = 3x3
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(32,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(64,(3,3))) #number of filters= 64
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(256))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(2))  # 2 classes for binary classification (gp and gn)
model.add(Activation("softmax"))
```

```
# Compile the model
model.compile(loss="categorical_crossentropy", optimizer="adam",
metrics=["accuracy"])

from keras.preprocessing.image import ImageDataGenerator

batch_size = 16

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Generating batches of augmented training data
train_generator = train_datagen.flow_from_directory(
    directory=train_path,
    target_size=(100, 100),
    batch_size=batch_size,
    class_mode="categorical"
)

# Generating batches of testing data
test_generator = test_datagen.flow_from_directory(
    directory=test_path,
    target_size=(100, 100),
    batch_size=batch_size,
    class_mode="categorical"
)

epochs = 100

history = model.fit(
    train_generator,
    epochs=epochs,
    steps_per_epoch=len(train_generator)
)



from keras.preprocessing.image import load_img
from keras.applications.vgg16 import preprocess_input
import numpy as np

# Load and preprocess the test image
test_image = load_img(r"C:\Users\ranga\Downloads\gpncc\gram_data\test\gp\21.jpg",
target_size=(100, 100))
test_image = img_to_array(test_image)
test_image = preprocess_input(test_image)
test_image = np.expand_dims(test_image, axis=0)

# Make predictions using the trained model
predictions = model.predict(test_image)

# Assuming you have class labels, you can map the predictions to class labels
class_labels = ["gn", "gp"]  # Replace with your actual class labels

# Get the predicted class index
predicted_class_index = np.argmax(predictions)

# Get the predicted class label
predicted_class_label = class_labels[predicted_class_index]
```

```
# Print the predicted class label
print("Predicted class:", predicted_class_label)
```