MYTHIC VISION: ILLUMINATING GODS THROUGH DEEP LEARNING

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE ENGINEERING

Submitted by

AVIRAL SINGH *(20BCE7373)*
ADITYA PATIL *(20BCE7013)*
KANAGIRI SUJAY ASHRITH *(20BCI7111)*
GITESH PRASHANT BHAVSAR (*20BCR7082)*

Under the Guidance of

PROF. TAUSEEF KHAN

SCHOOL OF COMPUTER SCIENCE ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

December 2023

CERTIFICATE

This is to certify that the Capstone Project work titled "**MYTHIC VISION: Illuminating Gods through Deep Learning**" that is being submitted by **Aviral Singh** ***(20BCE7373)***, **Aditya Patil** ***(20BCE7013)***, **Kanagiri Sujay Ashrith** ***(20BCI7111)*** and **Gitesh Prashant Bhavsar (*****20BCR7082)*** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

PROF. TAUSEEF KHAN
Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner I **Internal Examiner II**

Approved by

HoD, Name of the Department
School of Computer Science and Engineering

1. ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to the individuals who have been instrumental in the completion of this capstone project. Without their unwavering support, encouragement, and expertise, this endeavor would not have been possible. First and foremost, we extend our heartfelt thanks to our guide **PROF. TAUSEEF KHAN** for their guidance and invaluable insights throughout the entire process.

We would also like to acknowledge the SCOPE faculty for fostering an environment of learning and providing us with the necessary resources to undertake this project. The academic rigor and diverse perspectives have enriched our educational experience.

A Special Thanks to Dr MADHUSUDHANA RAO N, Dean Academics for including this capstone project in our curriculum which made it possible to learn and explore the best time of learning.

2. ABSTRACT

The "Mythic Vision" software is a capstone project aimed at enhancing the cultural experience of tourists in India. The project seeks to develop a deep learning-based software solution that can detect and classify representations of Indian deities present in cultural artifacts, sculptures, and artworks.

This software is tailored to find and classify these deity representations using smart algorithms and image recognition techniques. Its goal isn't just identifying these depictions but also telling the rich stories behind them, shedding light on Indian mythology's intricate tales. By providing tourists with insightful insights into Indian mythology, "Mythic Vision" aims to deepen their understanding of India's cultural heritage. It's about more than just observing; it invites travelers into a world where each depiction carries significant stories from centuries past.

Through this project, tourists can engage more deeply with India's cultural legacy, connecting with each deity's story. It's not just visiting tourist spots; it's an immersive journey that adds meaning to their travel experiences.

"Mythic Vision" isn't just a software, it's changing how tourists explore India's culture. By combining technology and culture, it's redefining how people connect with and understand cultural narratives during their travels.

3. TABLE OF CONTENTS

4. LIST OF TABLES & FIGURES

5. INTRODUCTION

Indian mythology is a treasure trove of ancient stories, intricate symbols, and divine tales. It has deeply influenced India's culture and spirituality, evident in its sculptures, artworks, and artifacts. However, this rich heritage hasn't seamlessly blended with modern technology. Our developed model, "Mythic Vision" steps in to bridge this gap. It aims to reveal the hidden stories in India's cultural artifacts, revolutionizing how people interact with Indian mythology during cultural tourism experiences.

Addressing challenges is central to the project's development. The absence of a comprehensive dataset of Indian deities poses a major hurdle. Selecting the most suitable deep learning model and deciding on prioritizing accuracy or a weighted decision-based approach present critical challenges. The application, MythicVision, aspires to blend Indian mythology with modern technology to enrich cultural tourism experiences. The project's approach involves creating unique datasets, selecting appropriate deep learning models, addressing accuracy challenges, and concluding with an intuitive application.

Following is the logo that we have designed for our software:

Fig. 1 Graphical representation of MythicVision logo.

6. BACKGROUND & LITERATURE SURVEY

In our search, we found no projects quite like "Mythic Vision," leaving a gap in Indian mythology-focused initiatives. While a Chinese project titled *"Traditional Chinese God Image Dataset: A Glimpse of Chinese Culture" [1]* celebrates their culture, there's been no equivalent focus on Indian mythology.

This makes "Mythic Vision" stand out, reshaping how tourists engage with India's rich culture. Bringing together deep learning and Indian mythology is crucial here. We're using technology to revolutionize how tourists experience India's cultural heritage.

7. PROBLEM DEFINITION

- Lack of Cultural Awareness: Tourists visiting India often lack awareness of the intricate stories and significance of Indian mythology, which is an integral part of the country's cultural heritage.

- Dependence on Guides: Presently, tourists seeking to learn about Indian mythology rely heavily on hiring guides, leading to financial expenses and the potential for inaccurate or misleading information.

- Language Barriers: Many tourists face language barriers when interacting with local guides, hindering their ability to fully understand and appreciate the nuances of Indian mythology.

- Absence of Quality Resources: The absence of easily accessible and reliable resources prevents tourists from independently exploring and understanding Indian mythology during their visits.

- Scarcity of Information: Due to the scarcity of comprehensive and accurate online resources regarding Indian deities, tourists remain uninformed about the deeper cultural context of their surroundings.

- Enhanced Cultural Interaction: Tourists are keen to engage with cultural narratives and historical tales to enhance their overall travel experience, making their interactions with India's cultural heritage more meaningful.

8. MOTIVATION

The motivation driving the "Mythic Vision" project is to bring together India's captivating stories and modern technology. We noticed that visitors often miss out on the rich tales behind artworks and sculptures. We're inspired to change that by using deep learning to uncover the hidden narratives of Indian deities. Our goal is to make these stories accessible to everyone, enhancing their understanding and enjoyment of cultural treasures. We believe that merging ancient tales with technology can make exploring culture more engaging. This project is fueled by the idea that when people connect with the stories, they can appreciate the artistry and meaning behind each piece even more. Our motivation pushes us to create the "Mythic Vision" software, offering a new way to explore culture and history.

9. OBJECTIVE OF THE PROPOSED MODEL

- The main goal of our project is to create a special application that can help people who visit India to learn about its ancient stories and gods more easily.

- Eliminate the need for tourists to hire expensive guides by offering a reliable and informative digital guide through the software.

- Enhance tourists' overall cultural experience in India by enabling them to connect more deeply with the history and stories behind the art and artifacts they encounter.

- Contribute to the well-being of local communities by ensuring that tourists receive accurate information and fostering a positive and educational tourism experience.

10. CONTRIBUTION

10.1 METHODOLOGIES

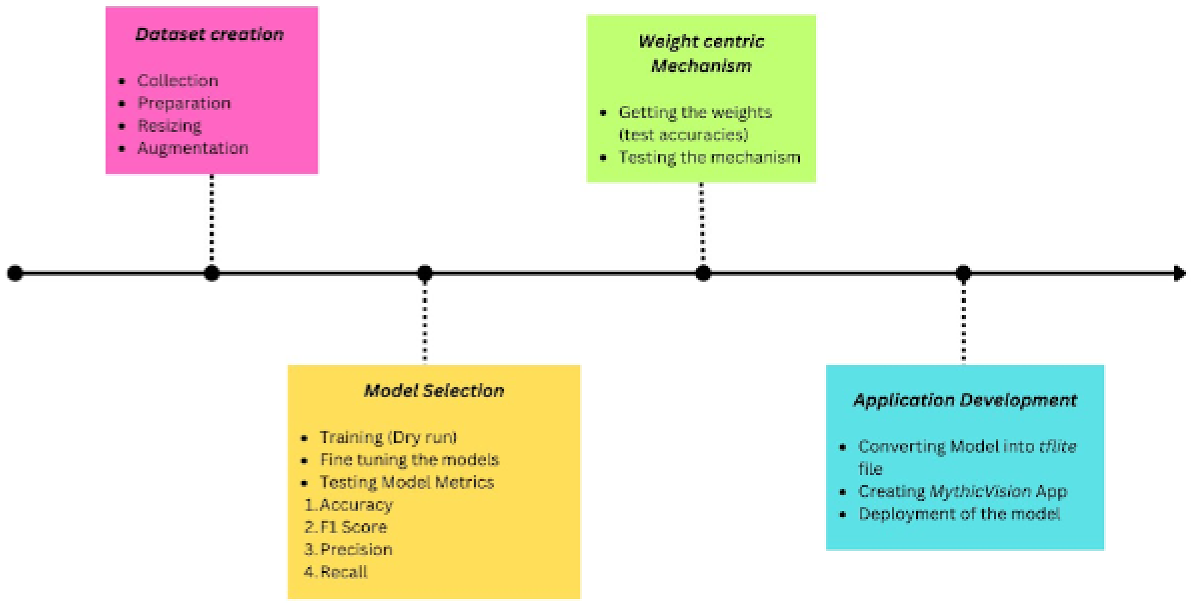


Fig. 2 Mind Map of the steps involved in the development of the project

The project unfolds in four key stages.

- First, it involves the process of sourcing, selecting, and curating images of Indian deities to construct our custom in-house dataset. The dataset and framework are publicly available on GitHub.
- Next, it focuses on the selection of appropriate deep learning models, such as MobileNet, ResNet, EfficientNet, and GoogleNet, based on their suitability for the project's objectives.
- A critical aspect of this project is the adoption of a weight-centric decision approach, prioritizing it over individual model accuracy to enhance the accuracy and reliability of the results.
- Finally, the project culminates in the development of an intuitive and user-friendly application that allows users to interact with and benefit from the project's findings.

10.2 IMAGE CLASSIFICATION FOR DEITIES

Class	Training set (Before augment.)	Training set (After augment.)	Validation set	Test set
Balaji	199	995	80	20
Durga Maa	200	1000	80	20
Ganesha	200	1000	80	20
Hanuman	199	995	80	20
Kali Maa	200	1000	80	20
Khatu Shyam	200	990	80	20
Krishna	200	1000	80	20
Sai Baba	200	995	80	20
Saraswati	200	1000	80	20
Shiva	199	995	80	20
Total	1997	9970	800	200

Table 1. . Brief outline of the experimental dataset along with its particulars

- The preparation of the dataset for "MythicVision" was a very important phase as there were no datasets present online. With a collection of ten distinct deities, each carrying unique symbolism, we downloaded a total of 600 images per deity from Google Images.

- From the initial pool, we started selecting images that will fulfill our requirements, we refined a total 300 images per deity that perfectly aligned with our project's objectives.

- Within this subset, we allocated 80 images for validation and 20 images were designated for testing. The remaining 200 images underwent a transformative process of augmentation. Through padding, we standardized the resolution of every image, establishing a consistent framework for analysis.

- Then each image underwent a rotation of 60 degrees and the image was selected as a new image. This approach allowed us to amplify the dataset significantly, transforming a single image into a collection of five. Consequently, 1000 images were deployed for each deity, ultimately forming the training phase. Following is the image classification.

10.3 MODEL TRAINING

This phase is the dataset deployment phase, we take the custom datasets, as discussed earlier, and subject them to comprehensive testing using four different deep learning models. The models that we are going to test our custom datasets with are:

- *MobileNetV2*
- *EfficientNetB0*
- *ResNet – 50*
- *GoogleNet Inception V3*

10.3.1 __EMPLOYED DEEP MODELS__

- ***__MobileNetV2__***

MobileNetV2 is designed to be lightweight and efficient, making it suitable for deployment on embedded systems. It balances model size, speed, and accuracy, making it a valuable architecture for various computer vision tasks, including image classification, object detection, and more. MobileNet V2 model has 53 convolution layers and 1 Average pool with nearly 350 GFLOP. The system comprises two main components. First, it includes the Inverted Residual Block, which plays a pivotal role in its architecture.

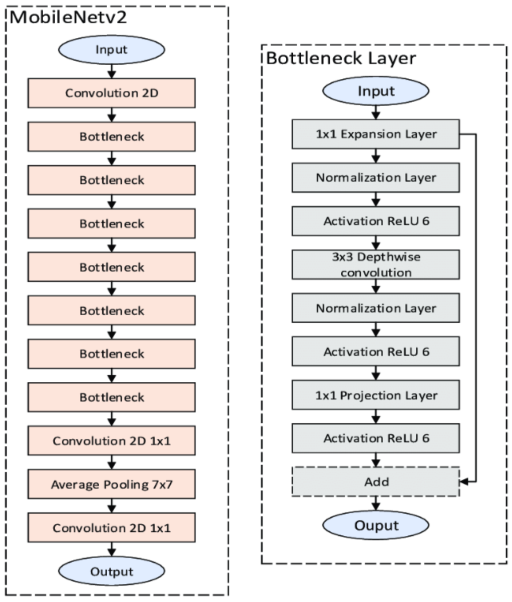
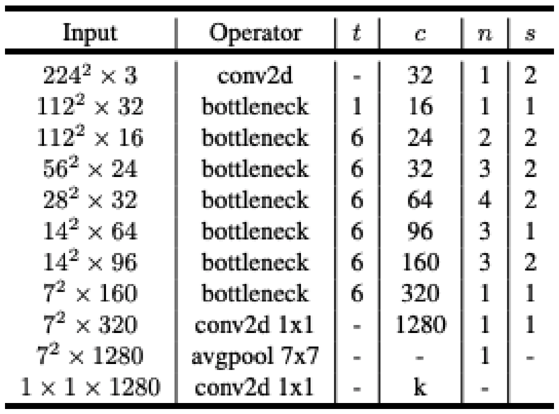


Fig. 3.a Main components of MobileNetv2



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Fig. 3.b Layer-wise architecture of MobileNetV2

Each block within the system consists of three distinct layers. Firstly, it includes a 1x1 Convolution layer with Relu6 activation, which serves as a non-linear transformation applied to the input data. Following this, the block incorporates a Depth wise Convolution layer, which involves the input independently for each channel, reducing computational complexity. Lastly, it features a 1x1 Convolution layer without any linearity, allowing the model to capture linear relationships in the data as shown in the above figure.

- ***EfficientNetB0***

EfficientNetB0 is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient. The EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. EfficientNet employs a compound scaling approach, adjusting the width, depth, and resolution of the network with a compounding coefficient (ϕ). The key elements include:

- Width Scaling (w): It increases the number of channels in convolutional layers, effectively widening the network.
- Depth Scaling (d): This increases the number of layers, making the network deeper.
- Resolution Scaling (r): It changes the input image's resolution, allowing the network to handle different image sizes.

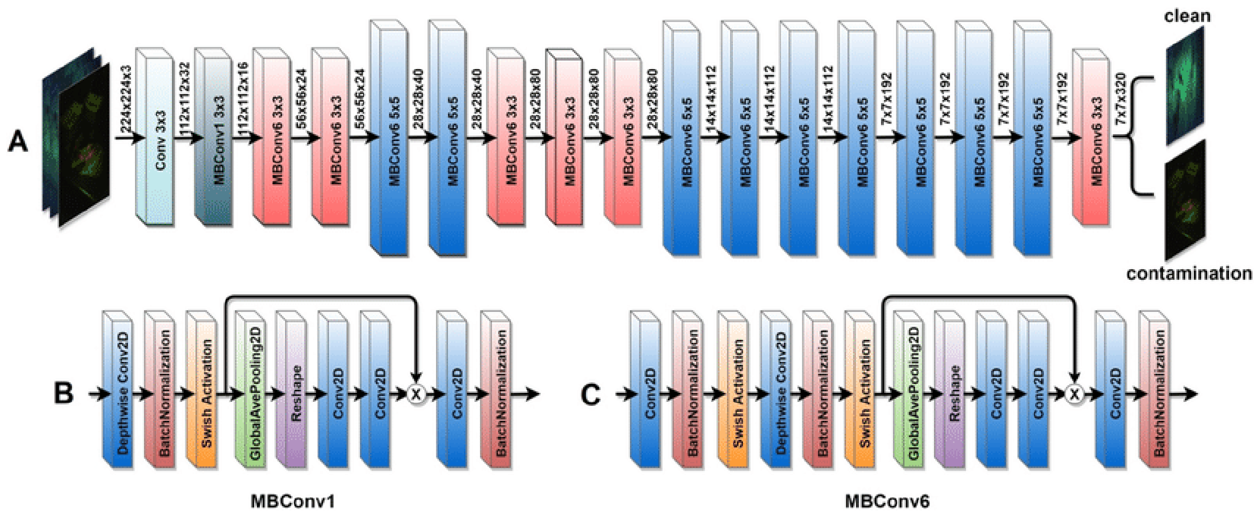


Fig. 4. (A) Layer wise representation of the EfficientNet-B0 model. **(B)** The building blocks of MBConv1. **(C)** The building blocks of MBConv6.

EfficientNetB0 consists of an initial 3x3 convolutional layer, followed by a stem block for feature extraction. The core architecture comprises stacked blocks, each containing MobileNetV2-like inverted residual structures (MBConv). These MBConv blocks consist of depthwise separable convolutions, batch normalization, and ReLU activation functions. The model incorporates scaling factors (α, β, γ) to adjust depth, width, and resolution. The final

layers include global average pooling, fully connected layers, and an output layer for predictions as shown in the above figure.
EfficientNetB0, part of the EfficientNet family of models, represents a breakthrough in neural network architecture optimization. Developed to provide superior performance in terms of accuracy and efficiency, EfficientNetB0 achieves state-of-the-art results in image classification tasks. Its architecture incorporates a novel compound scaling method that uniformly scales the network's depth, width, and resolution, optimizing the model for diverse computational resources. With a focus on balancing accuracy and computational cost, EfficientNetB0 stands as a versatile and scalable choice, making it well-suited for various applications ranging from mobile devices to resource-constrained environments.
EfficientNetB0's innovative scaling strategy ensures exceptional performance across a spectrum of tasks, making it a compelling choice for applications where computational efficiency is paramount.

- ***ResNet – 50***

ResNet-50 is a deep neural network architecture consisting of 48 convolutional layers, 1 average pooling layer, and 1 fully connected layer. It is a type of deep residual neural network that utilizes 3-layered bottleneck blocks to facilitate feature extraction and model training. In the network, the Rectified Linear Unit (ReLU) activation function is applied to every set of three consecutive convolutional layers. This non-linear activation helps the network capture intricate features within the data. Additionally, batch normalization techniques are incorporated to ensure stable and accurate training. ResNet-50 is a computationally intensive model, with approximately 3.8 billion Floating Point Operations (FLOPs). This high level of computational complexity is one of the reasons behind its remarkable performance in various computer vision tasks, particularly image classification. The resnet50 architecture comprises the below mentioned layers as depicted in the following figure.

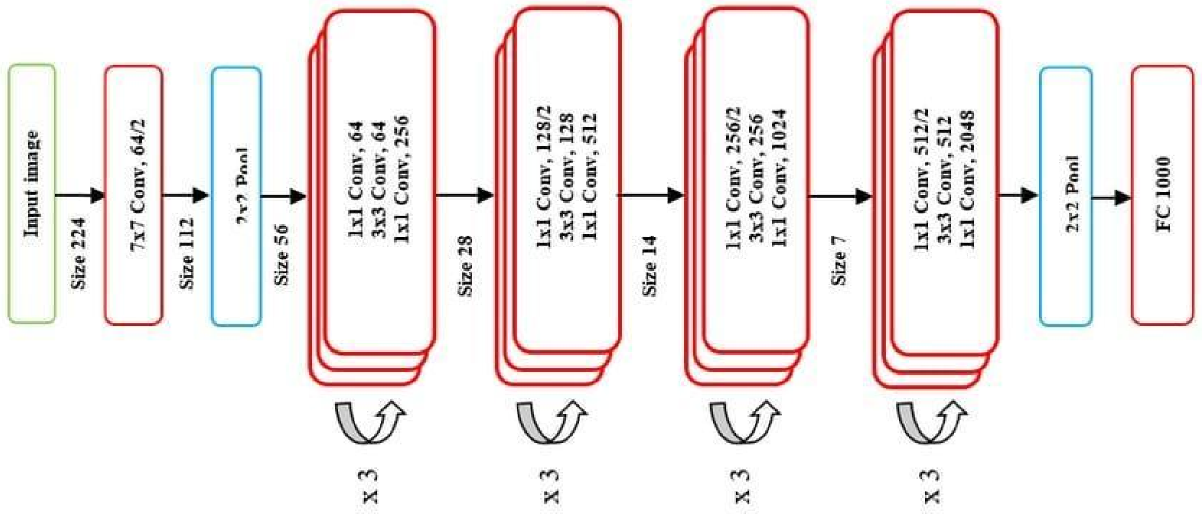


Fig. 5 Architecture of Layers of ResNet-50

The network architecture consists of various convolutional layers with different kernel sizes and numbers of channels. The first layer employs 64 kernels with a size of 7 * 7 and a stride of 2, resulting in one layer. Subsequently, a max-pooling layer follows with a stride of 2.
In the subsequent convolutional layers, there are different kernel sizes, including 1 * 1 with 64 channels, 3 * 3 with 64 channels, and 1 * 1 with 256 channels. This set of layers is repeated three times, totaling nine layers.
Another set of convolutional layers is composed of 1 * 1 with 128 channels, 3 * 3 with 128 channels, and 1 * 1 with 512 channels. This set is repeated four times, giving us 12 layers.
Following that, there are convolutional layers with kernel sizes of 1 * 1 and channels of 256, 3 * 3 and channels of 256, and 1 * 1 with 1024 channels. This group is repeated six times, leading to 18 layers.
Additionally, there are convolutional layers comprising 1 * 1 with 512 channels, 3 * 3 with 512 channels, and 1 * 1 with 2048 channels. This set is repeated three times, resulting in nine layers.
After these convolutional layers, average pooling is applied, followed by a fully connected layer containing 1000 nodes and a SoftMax function, which amounts to one layer.

- ***GoogleNet Inception V3***

The Google Inception V3, also known as Inception Net version 3, is a deep convolutional neural network architecture. It builds upon the foundation laid by its predecessors, Inception V1 and Inception V2, by adding more layers to the network. The Inception V3 model comprises a total of 42 layers. InceptionV3, a component of the GoogleNet architecture, comprises an initial stem block for feature extraction, followed by a series of diverse Inception modules. These modules leverage parallel paths with different kernel sizes, including 1x1, 3x3, and 5x5 convolutions, as well as pooling operations, as depicted in Fig.6, to capture multi-scale features efficiently. Reduction blocks are employed to decrease spatial dimensions, and auxiliary classifiers aid in mitigating the vanishing gradient problem during training. The architecture concludes with global average pooling, fully connected layers, and a SoftMax output layer for classification. Following are the Architecture and Components of the Inception V3 model.

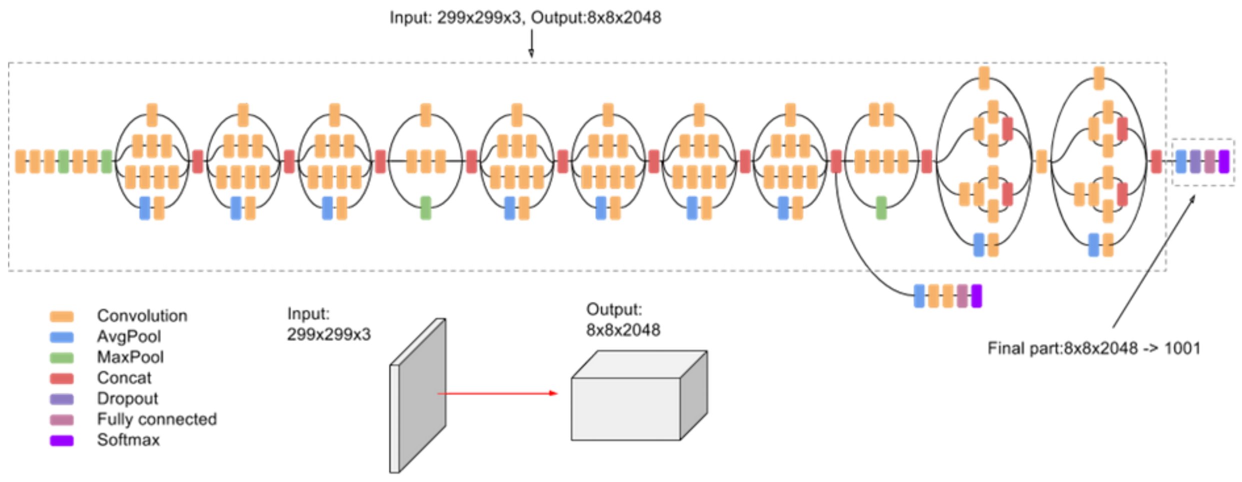
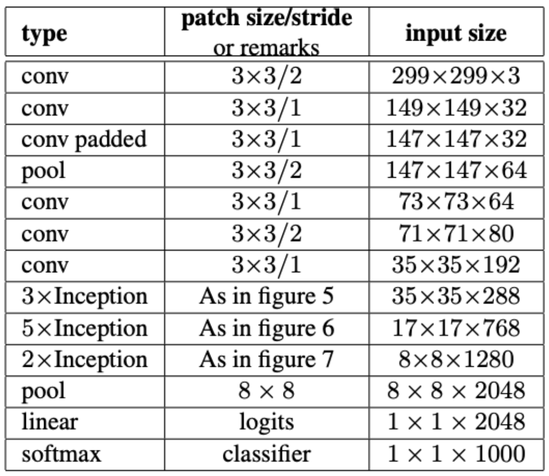


Fig.6.a Architecture of GoogleNet (Inception V3)

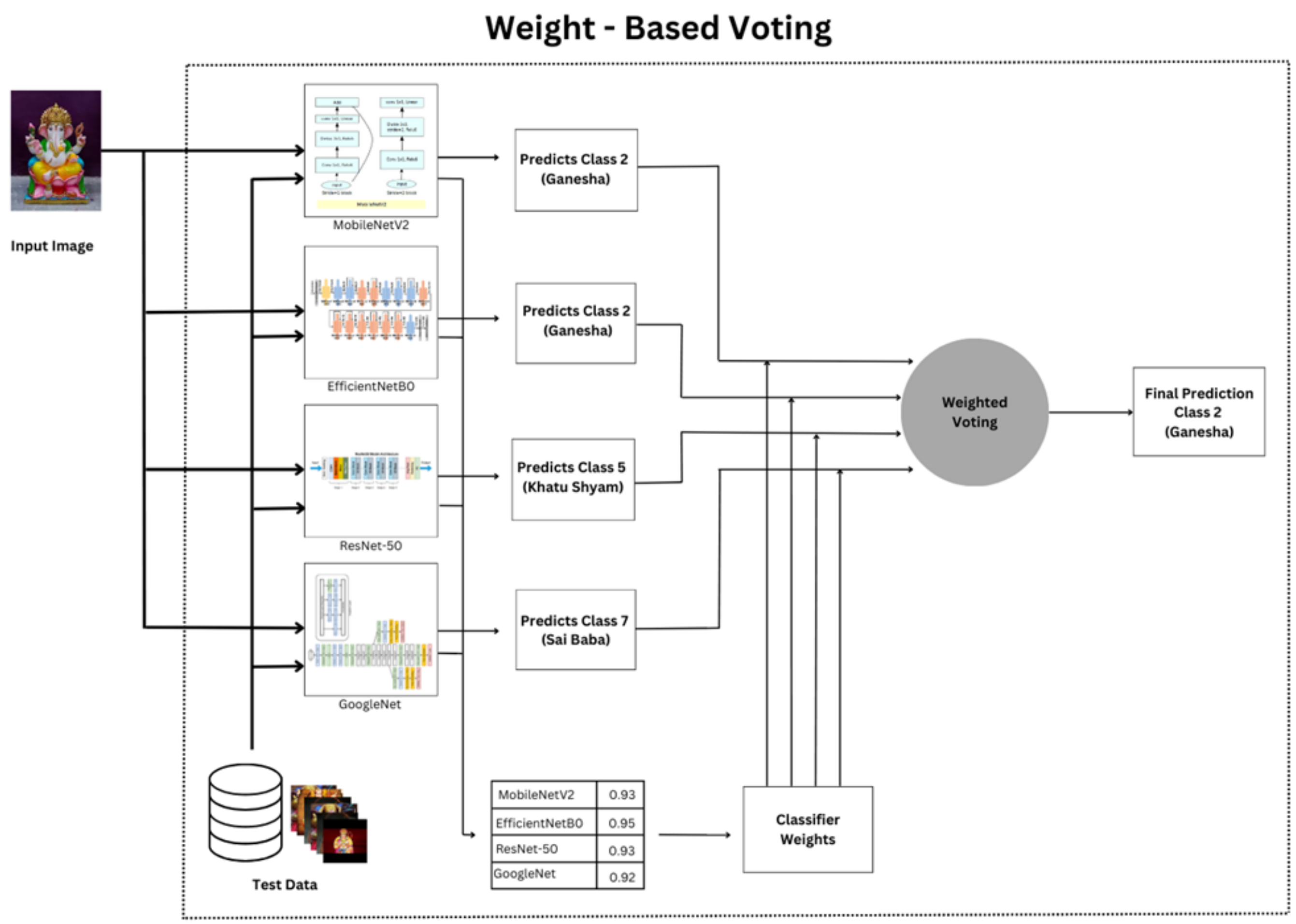


type	patch size/stride or remarks	input size
conv	$3{\times}3/2$	$299{\times}299{\times}3$
conv	$3{\times}3/1$	$149{\times}149{\times}32$
conv padded	$3{\times}3/1$	$147{\times}147{\times}32$
pool	$3{\times}3/2$	$147{\times}147{\times}64$
conv	$3{\times}3/1$	$73{\times}73{\times}64$
conv	$3{\times}3/2$	$71{\times}71{\times}80$
conv	$3{\times}3/1$	$35{\times}35{\times}192$
$3{\times}$Inception	As in figure 5	$35{\times}35{\times}288$
$5{\times}$Inception	As in figure 6	$17{\times}17{\times}768$
$2{\times}$Inception	As in figure 7	$8{\times}8{\times}1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

Fig.6.b Components of the Inception V3 model

10.4 WEIGHT-CENTRIC DECISION MECHANISM.

To address the complexities of the majority-based voting approach and ensure more accurate results, the project integrates a weight-centric decision mechanism, which will be elaborated upon in the subsequent paragraph. This approach enhances the reliability and precision of the desired result, further improving the software's performance.
In the Weight-Based Voting mechanism, the project employs a strategic approach to enhance classification accuracy. Prior to making predictions, each of the four models undergo thorough testing on a dedicated dataset, allowing for the calculation of individual model accuracies. These accuracies are used as weight values for the final classification. For example, if, after testing, the model accuracies are found to be 95%, 93%, 93%, and 92%, and an input image is analyzed by the models, the predictions are as follows: Ganesha – 2, Hanuman – 1, Khatu Shyam – 1.
Weight values are then assigned to each class based on their respective model accuracies. For instance, Ganesha is assigned a weight value of 0.95 + 0.93, summing up to 1.88, while Hanuman receives a weight value of 0.93, and Khatu Shyam gets 0.92. With class Ganesha having the highest weighted value, the final prediction becomes Ganesha.


Weight - Based Voting
Input Image
MobileNetV2
EfficientNetB0
ResNet-50
GoogleNet
Predicts Class 2 (Ganesha)
Predicts Class 2 (Ganesha)
Predicts Class 5 (Khatu Shyam)
Predicts Class 7 (Sai Baba)
Weighted Voting
Final Prediction Class 2 (Ganesha)
Test Data
Classifier Weights

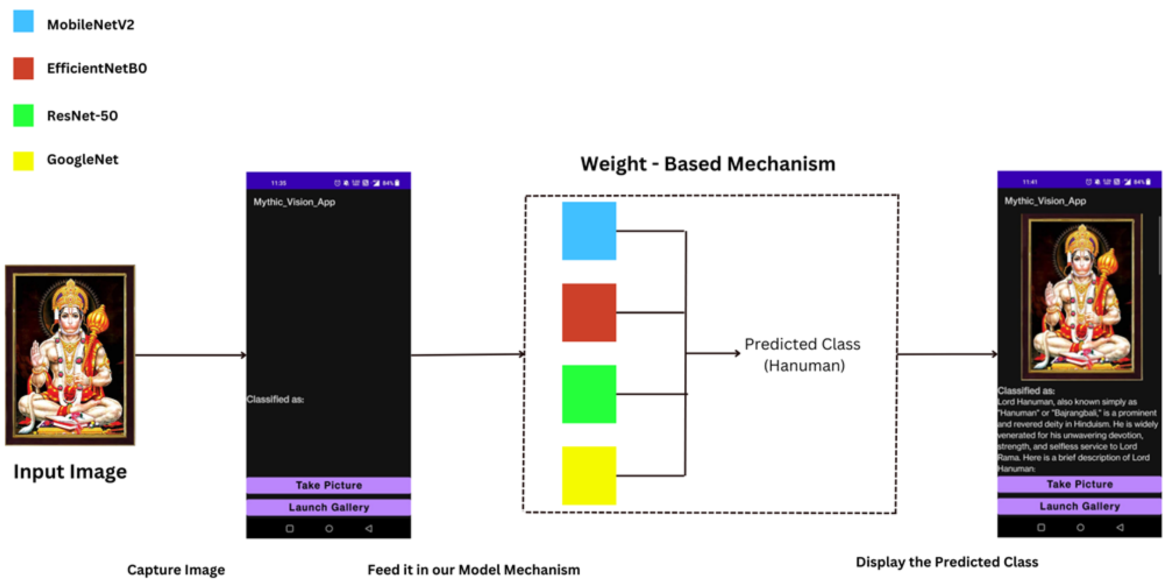
MobileNetV2	0.93
EfficientNetB0	0.95
ResNet-50	0.93
GoogleNet	0.92

Fig.7 Weight-centric decision mechanism.

10.5 DEVELOPMENT OF *MythicVision* APPLICATION

In this section, the operational flow of the developed application is being illustrated. The final model, which was created after the key stages of dataset creation, model selection, and the weight-based approach, is seamlessly integrated into an application. This is achieved through the conversion of the model into a TensorFlow Lite file, which is then exported to Android Studio, a dynamic platform for application development, where a user-friendly application with a basic ui interface was developed . Within this application, users encounter an interface featuring two essential buttons. The "Take Pictures" button activates the device's camera, enabling users to capture real-life images that align with their exploration of Indian mythology. The "Launch Gallery" button provides a convenient portal for users to access their device's image gallery and select a picture they may have captured previously.

The user initiates the process by capturing an image through the app. This image is subsequently channeled into the integrated models, where the deep learning models process it, where the weight-based approach is thoroughly applied. The developed application then presents the user with the predicted class corresponding to the captured image, along with a comprehensive information related to the recognized deity.



Classified as:
Lord Hanuman, also known simply as "Hanuman" or "Bajrangbali," is a prominent and revered deity in Hinduism. He is widely venerated for his unwavering devotion, strength, and selfless service to Lord Rama. Here is a brief description of Lord Hanuman:

Fig.8 Mechanism of Application usage

11. FUTURE SCOPE

- Multilingual Support: Extend the software to provide information in various languages to cater to a diverse range of tourists.

- Collaborative Platform: Create a community-driven platform where users can contribute additional information and stories about detected deities.

- Expansion to Other Cultures: Adapt the software to detect and classify deities from different cultures around the world.

- Language Diversity: Expand the software to offer information in different languages, making it accessible to a wider range of tourists from around the world.

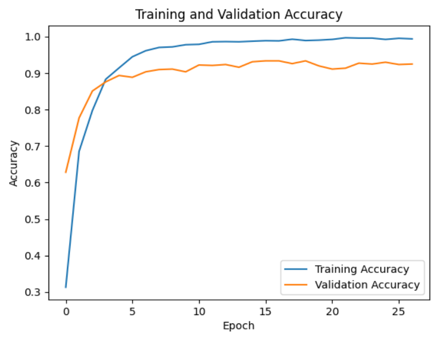
- Interactive Quizzes: Integrate interactive quizzes and games within the software to make learning about Indian mythology even more engaging and fun.

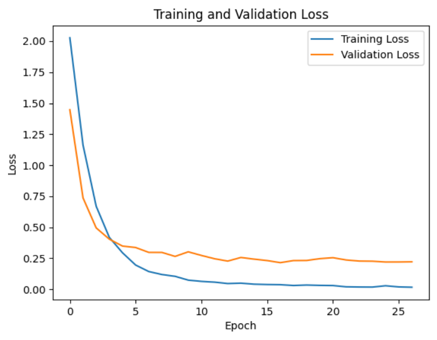
12. Experimental Results and Analysis

A series of experiments was performed on an in-house dataset. These experiments employed various deep learning models, and the resulting outcomes have been documented. Subsequently, an insightful analysis and discussion are presented based on the obtained results.

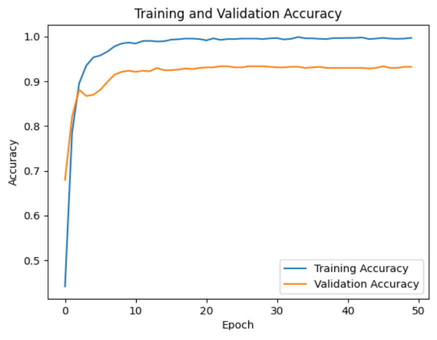
12.1 Accuracy and Loss Plots

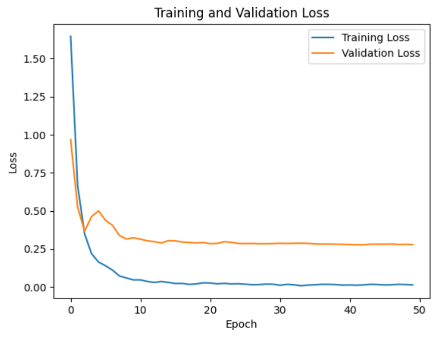
Following is the presentation of accuracy and loss trends observed during the training of the model. Specifically, a total of four accuracy plots and four loss plots corresponding to four distinct models MobileNetV2, EfficientNetB0, Resnet -50, GoogleNet are shown.



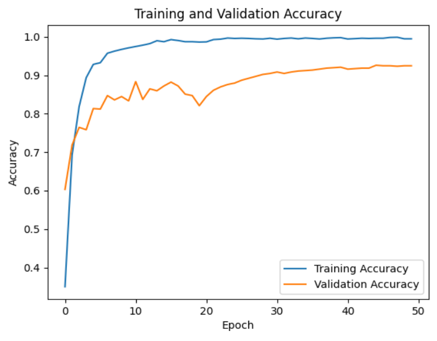


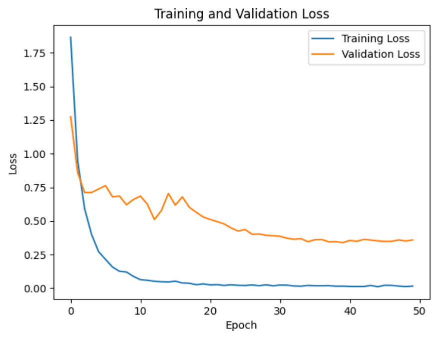
(a) (b)



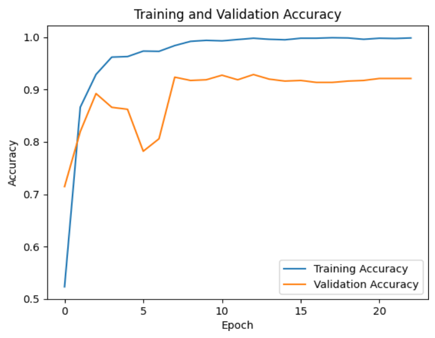


(c) (d)

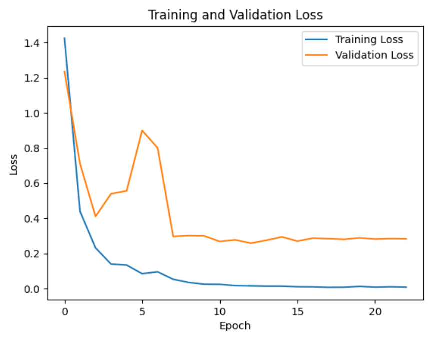




(e) (f)



(g)

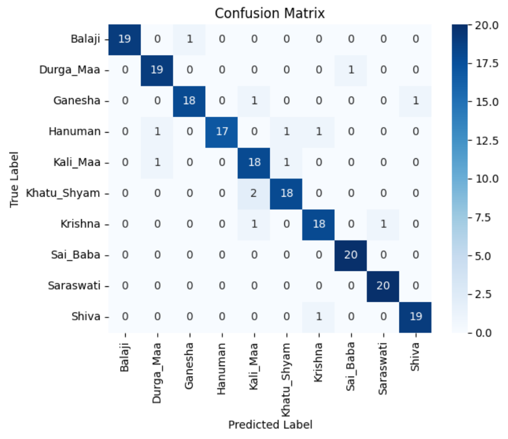


(h)

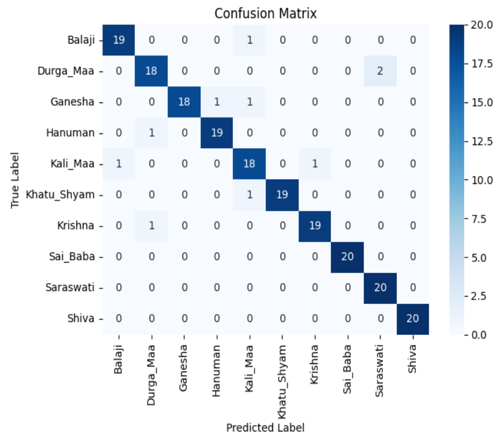
Fig.9 Graphical representation of epoch-wise training accuracy and loss of different deep models. (a-b) Corresponding training accuracy and loss plot for MobileNetV2, (c-d) Training accuracy and loss plot for EfficientNetB0, (e-f) training accuracy and loss plot for ResNet-50, (g-h) same plot for GoogleNet (Inception V3).

12.2 Confusion Matrix

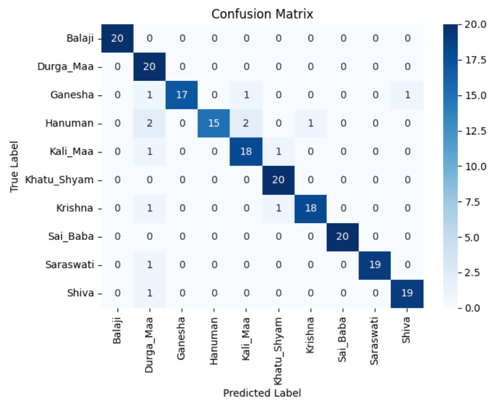
Next, the confusion matrix is shown. The confusion matrix is important as it offers a detailed breakdown of a model's predictions, allowing for a deeper analysis of errors, model fine-tuning, and the selection of appropriate evaluation metrics.



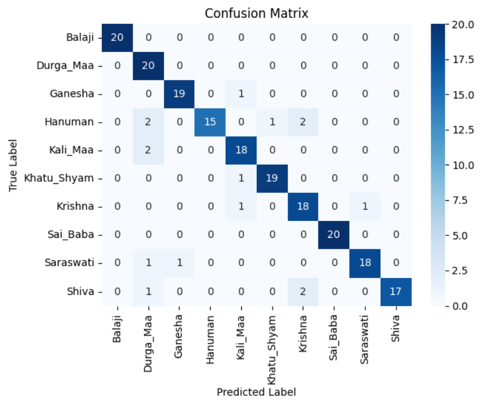
(a)



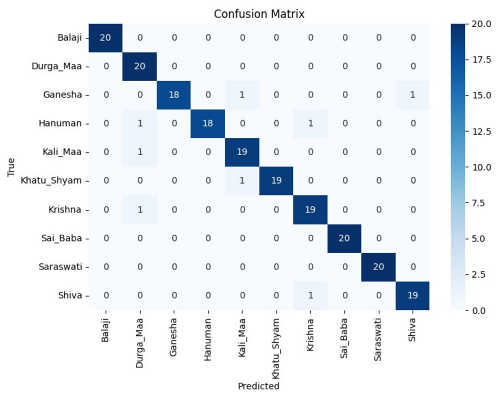
(b)



(c)



(d)

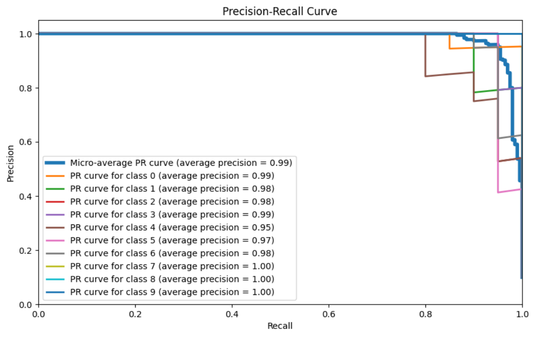


(e)

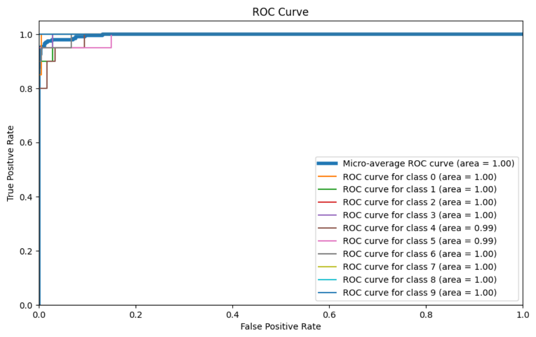
Fig. 10. Generated Confusion matrix of different deep models after classification. (a-d) Confusion matrix of MobileNetV2, EfficientNetB0, ResNet-50 and GoogleNet, (e) generate matrix using weight-centric decision mechanism.

12.3 Precision and Recall curve

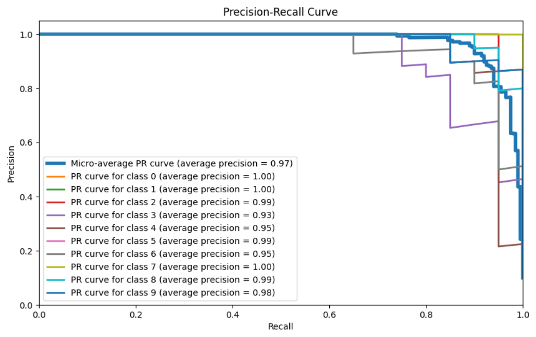
Precision-Recall (PR) and ROC curves offer insights beyond traditional accuracy metrics. They allow for a more nuanced evaluation of model performance, especially in situations involving imbalanced datasets, varying error costs, or a need to fine-tune classification thresholds. These curves aid in making informed decisions about model selection, threshold adjustment, and optimizing performance for specific applications [24]. The precision-recall curve illustrates the trade-off between precision and recall. The ROC curve showcases the model's true positive rate against the false positive rate.



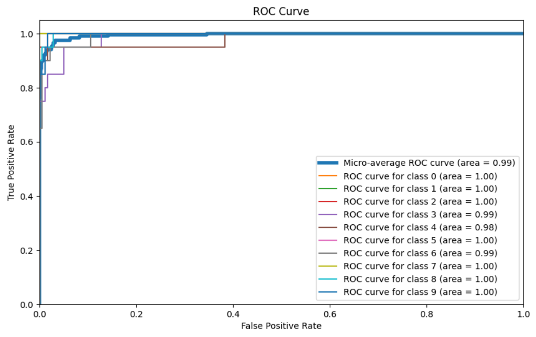
(a)



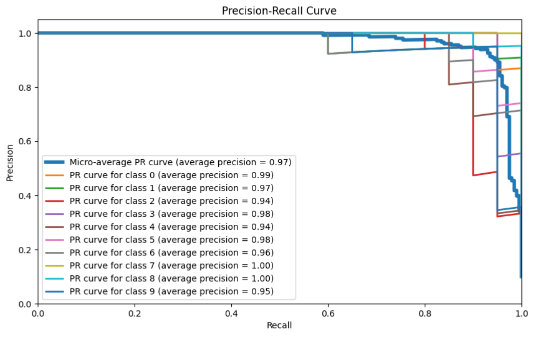
(b)



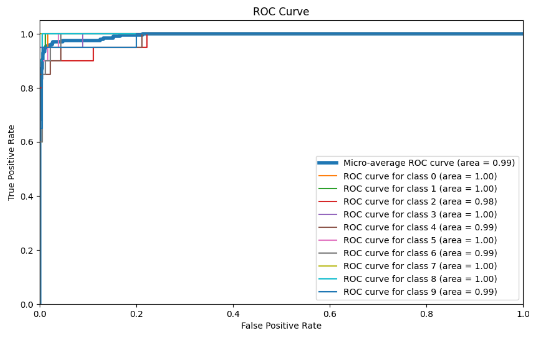
(c)



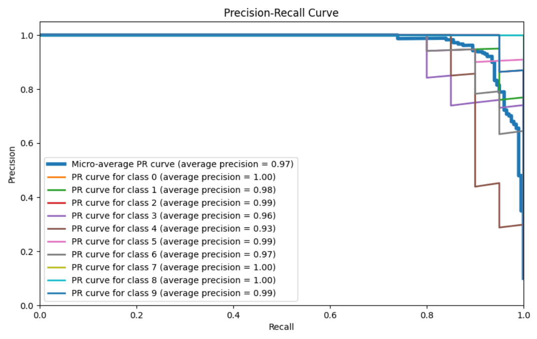
(d)



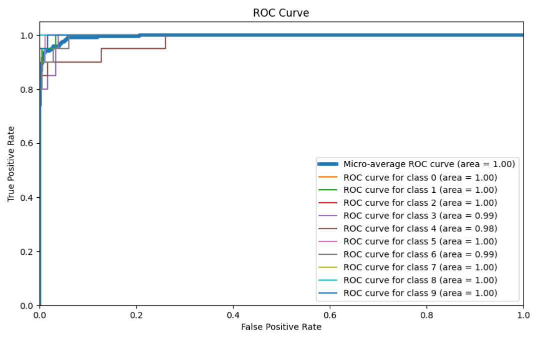
(e)



(f)



(g)



(h)

Fig.11 Precision(P)-Recall(R) curve and ROC curve of different deep models. (a-b) Corresponding PR and ROC curve for MobileNetV2, (c-d) PR and ROC curve for EfficientNetB0, (e-f) PR and ROC curve for ResNet-50, (g-h) same plot for GoogleNet (Inception V3).

12.4 __Performance metrics of deep learning models and weight-centric mechanism.__

Our evaluation focuses on attaining Test Accuracy, Test Loss, F1 Score, Precision and Recall. Metrics like F1 score, precision, and recall are crucial in this paper as they provide a balanced assessment of a model's performance in situations involving imbalanced datasets or varying costs of errors. Unlike accuracy, these metrics help evaluate the model's ability to correctly classify both positive and negative instances and distinguish between types of errors, making them valuable in real-world applications.

Employed Deep Models	CA	F1 Score	P	R	Error
MobileNetV2	0.93	0.9302	0.9325	0.93	0.07
Efficient Net B0	0.95	0.9502	0.9516	0.95	0.05
ResNet-50	0.93	0.9305	0.9404	0.93	0.07
GoogleNet (Inception V3)	0.92	0.9204	0.9292	0.92	0.08
Weight centric mechanism	0.96	0.9602	0.9629	0.96	0.04

Table 2. summary of the performance metrics for various deep learning models and the weight-centric mechanism.

12.5 <u>Performance comparison of deep models and weight centric mechanism.</u>

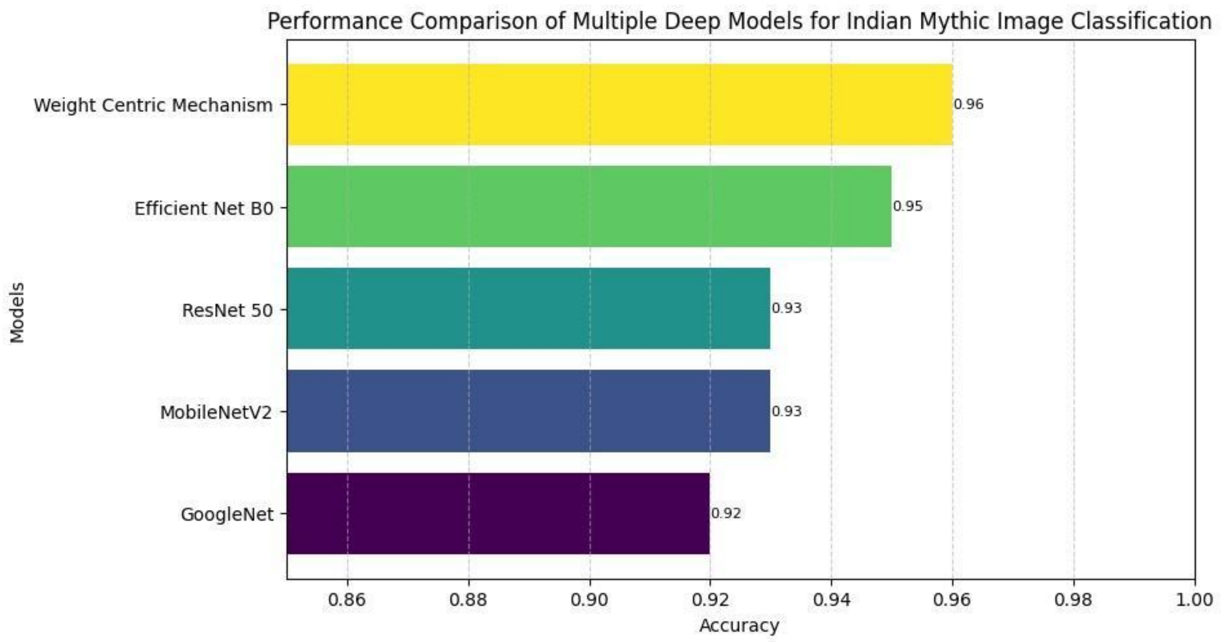


Fig.12 Comparison of MobileNetV2, EfficientNetB0, ResNet-50, GoogleNet (Inception V3), and Weight centric mechanism.

13. CONCLUSION

In conclusion, the "Mythic Vision" software represents a significant advancement at the crossroads of modern deep learning and India's rich mythology. Its impact on cultural exploration will be remarkable as tourists will use this tool, they will actively engage with Indian mythology, gaining a deeper understanding of the culture. This infusion of knowledge will help the travelers form meaningful connections with the culture they're exploring. They move beyond surface-level experiences and gain insights into the symbolism, stories, and traditions that shape India's cultural landscape. This deeper engagement ensures that tourists leave not only with memories but with a genuine appreciation of the heritage they've encountered. The MythicVision application is more than just a tool; it's a catalyst for an educational tourism experience. It's set to enhance India's global appeal as a tourist destination, offering a richer and more profound journey for all. The fusion of deep learning and cultural exploration promises to elevate India's tourism industry, providing a more enlightening experience for visitors.

14. APPENDIX

14.1 Codes for image resizing with padding:

```
from PIL import Image
import os


def resize_images(source_dir, dest_dir, target_size):
    if not os.path.exists(dest_dir):
        os.makedirs(dest_dir)

    image_files = [f for f in os.listdir(source_dir) if f.lower().endswith(('.jpg', '.png', '.jpeg'))]

    for image_file in image_files:
        image_path = os.path.join(source_dir, image_file)
        dest_path = os.path.join(dest_dir, image_file)

        # Open the image and calculate the aspect ratio
        img = Image.open(image_path)
        width, height = img.size
        aspect_ratio = width / height

        # Resize the image while maintaining the aspect ratio
        new_width = target_size[0]
        new_height = int(new_width / aspect_ratio)
        resized_img = img.resize((new_width, new_height), Image.LANCZOS )

        # Create a blank canvas of the target size and paste the resized image
        canvas = Image.new("RGB", target_size, "black")
        paste_x = 0
        paste_y = (target_size[1] - new_height) // 2
        canvas.paste(resized_img, (paste_x, paste_y))

        # Save the resized image with proper aspect ratio
        canvas.save(dest_path)
```

```
# Specify paths and target size
source_directory = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\training\Shiva'
destination_directory = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\dataset\training\Shiva'
target_size = (224, 224)  # The desired target size

# Resize all images in the source directory and save them to the destination directory
resize_images(source_directory, destination_directory, target_size)
```

14.2 Codes for Data Augmentation:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
from PIL import Image, ImageOps  # Add this line to import ImageOps module

# Update the directory paths using raw strings or escaped backslashes
input_dir = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\dataset\training\Shiva'
output_dir = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\dataset\training_augmented\Shiva'

if not os.path.exists(output_dir):
   os.makedirs(output_dir)

# Data augmentation
num_augmented_images = 5  # Number of augmented images per input image

# Define data augmentation parameters
data_gen = ImageDataGenerator(
   rescale=1.0 / 255,
   rotation_range=20,
   width_shift_range=0.2,
   height_shift_range=0.2,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True,
   fill_mode='nearest'
)

# Load original images
image_files = [os.path.join(input_dir, filename) for filename in os.listdir(input_dir) if
            filename.endswith(".jpg") or filename.endswith(".png") or filename.endswith(".jpeg")]

for img_path in image_files:
   img = tf.keras.preprocessing.image.load_img(img_path, target_size=(224, 224))
   img_array = tf.keras.preprocessing.image.img_to_array(img)
   img_array = img_array.reshape((1,) + img_array.shape)  # Add batch dimension

   # Generate augmented images using the flow method
   img_name = os.path.basename(img_path).split('.')[0]  # Extract image name without extension
   generator = data_gen.flow(img_array, batch_size=1)

   for i in range(num_augmented_images):
       batch = generator.next()
       augmented_img = batch[0]

       # Calculate padding
       width_diff = 224 - augmented_img.shape[1]
       height_diff = 224 - augmented_img.shape[0]
       left_padding = width_diff // 2
       top_padding = height_diff // 2

       # Apply padding with black color
       padded_img = ImageOps.expand(Image.fromarray((augmented_img * 255).astype('uint8')),
                                   (left_padding, top_padding, width_diff - left_padding, height_diff -
top_padding),
                            fill='black')

       # Save the padded augmented image
       augmented_img_path = os.path.join(output_dir, f"{img_name}aug{i}.jpg")
       padded_img.save(augmented_img_path)

print(f"{len(image_files) * num_augmented_images} augmented images with proper padding
generated and saved.")
```

14.3 Codes used for training data models and their training accuracy and validation loss plot:

- ***MobileNetV2***

```
import numpy as np
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt


train_dir = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\dataset\training'
validation_dir = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\dataset\validation'
test_dir = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\dataset\testing'

datagen = ImageDataGenerator(
  rotation_range=20,
  width_shift_range=0.2,
  height_shift_range=0.2,
  zoom_range=0.2,
  fill_mode='nearest'
)

def load_and_preprocess_image(file_path):
  img = load_img(file_path, target_size=(224, 224))
  img_array = img_to_array(img)
```

```
    img_array = preprocess_input(img_array)
    return img_array


def load_data_from_directory(directory):
    data = []
    labels = []
    class_indices = {}
    for idx, class_name in enumerate(sorted(os.listdir(directory))):
        class_indices[class_name] = idx
        class_dir = os.path.join(directory, class_name)
        for filename in os.listdir(class_dir):
            if filename.lower().endswith((".jpg", ".jpeg", ".png")):  # Supporting multiple image
formats
                img_path = os.path.join(class_dir, filename)
                img_array = load_and_preprocess_image(img_path)
                data.append(img_array)
                labels.append(idx)
    return np.array(data), np.array(labels), class_indices


train_data, train_labels, class_indices = load_data_from_directory(train_dir)
validation_data, validation_labels, _ = load_data_from_directory(validation_dir)
test_data, test_labels, _ = load_data_from_directory(test_dir)

# Convert labels to categorical format
num_classes = len(class_indices)
train_labels = to_categorical(train_labels, num_classes)
validation_labels = to_categorical(validation_labels, num_classes)
test_labels = to_categorical(test_labels, num_classes)


# Create the MobileNetV2 model with custom classification layers
base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),    loss='categorical_crossentropy',
metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True,
verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=4, min_lr=1e-6)

# Train the model
batch_size = 32
history = model.fit(
   datagen.flow(train_data, train_labels, batch_size=batch_size),
   epochs=50,
   validation_data=(validation_data, validation_labels),
   callbacks=[checkpoint, early_stopping, reduce_lr]
)


# training and validation accuracy plot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('accuracy_plot.png')
plt.close()


# training and validation loss plot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('loss_plot.png')
plt.close()



results_folder = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Results'
if not os.path.exists(results_folder):
    os.makedirs(results_folder)
os.rename('accuracy_plot.png', os.path.join(results_folder, 'accuracy_plot.png'))
os.rename('loss_plot.png', os.path.join(results_folder, 'loss_plot.png'))
```

- ***EfficientNetB0***

```
import numpy as np
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications.efficientnet import preprocess_input as
efficientnet_preprocess_input
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt


train_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\training'
validation_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\validation'
test_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\testing'

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    fill_mode='nearest'
)

def load_and_preprocess_image(file_path):
    img = load_img(file_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = efficientnet_preprocess_input(img_array)
    return img_array


def load_data_from_directory(directory):
    data = []
    labels = []
    class_indices = {}
    for idx, class_name in enumerate(sorted(os.listdir(directory))):
        class_indices[class_name] = idx
        class_dir = os.path.join(directory, class_name)
        for filename in os.listdir(class_dir):
            if filename.lower().endswith((".jpg", ".jpeg", ".png")):  # Supporting multiple image
formats
                img_path = os.path.join(class_dir, filename)
                img_array = load_and_preprocess_image(img_path)
                data.append(img_array)
                labels.append(idx)
    return np.array(data), np.array(labels), class_indices


train_data, train_labels, class_indices = load_data_from_directory(train_dir)
validation_data, validation_labels, _ = load_data_from_directory(validation_dir)
test_data, test_labels, _ = load_data_from_directory(test_dir)

# Convert labels to categorical format
num_classes = len(class_indices)
train_labels = to_categorical(train_labels, num_classes)
validation_labels = to_categorical(validation_labels, num_classes)
test_labels = to_categorical(test_labels, num_classes)
```

```
base_model = EfficientNetB0(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])



# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

for layer in base_model.layers:
    layer.trainable = False  # Freeze all layers

# Callbacks
checkpoint = ModelCheckpoint('best_model_EfficientNet.h5', monitor='val_loss', save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=4, min_lr=1e-6)

# Train the model
batch_size = 32
history = model.fit(
    datagen.flow(train_data, train_labels, batch_size=batch_size),
    epochs=50,
    validation_data=(validation_data, validation_labels),
    callbacks=[checkpoint, early_stopping, reduce_lr]
)


# training and validation accuracy plot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('accuracy_plot.png')
plt.close()


# training and validation loss plot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('loss_plot.png')
plt.close()



results_folder = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Results'
if not os.path.exists(results_folder):
    os.makedirs(results_folder)
os.rename('accuracy_plot.png', os.path.join(results_folder, 'accuracy_plot.png'))
os.rename('loss_plot.png', os.path.join(results_folder, 'loss_plot.png'))
```

- ***ResNet-50***

```
import numpy as np
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications.resnet import preprocess_input
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
```

```
train_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\training'
validation_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\validation'
test_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\testing'

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    fill_mode='nearest'
)

def load_and_preprocess_image(file_path):
    img = load_img(file_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = preprocess_input(img_array)
    return img_array


def load_data_from_directory(directory):
    data = []
    labels = []
    class_indices = {}
    for idx, class_name in enumerate(sorted(os.listdir(directory))):
        class_indices[class_name] = idx
        class_dir = os.path.join(directory, class_name)
        for filename in os.listdir(class_dir):
            if filename.lower().endswith((".jpg", ".jpeg", ".png")):  # Supporting multiple image
formats
                img_path = os.path.join(class_dir, filename)
                img_array = load_and_preprocess_image(img_path)
                data.append(img_array)
                labels.append(idx)
    return np.array(data), np.array(labels), class_indices
```

```
train_data, train_labels, class_indices = load_data_from_directory(train_dir)
validation_data, validation_labels, _ = load_data_from_directory(validation_dir)
test_data, test_labels, _ = load_data_from_directory(test_dir)

# Convert labels to categorical format
num_classes = len(class_indices)
train_labels = to_categorical(train_labels, num_classes)
validation_labels = to_categorical(validation_labels, num_classes)
test_labels = to_categorical(test_labels, num_classes)


# Create the MobileNetV2 model with custom classification layers
base_model = ResNet50(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint('best_model_ResNet.h5', monitor='val_loss',
save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=4, min_lr=1e-7)

# Train the model
batch_size = 32
history = model.fit(
    datagen.flow(train_data, train_labels, batch_size=batch_size),
    epochs=50,
    validation_data=(validation_data, validation_labels),
    callbacks=[checkpoint, early_stopping, reduce_lr]
)


# training and validation accuracy plot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('accuracy_plot.png')
plt.close()


# training and validation loss plot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('loss_plot.png')
plt.close()



results_folder = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Results'
if not os.path.exists(results_folder):
    os.makedirs(results_folder)
os.rename('accuracy_plot.png', os.path.join(results_folder, 'accuracy_plot.png'))
os.rename('loss_plot.png', os.path.join(results_folder, 'loss_plot.png'))
```

- ***<u>GoogleNet Inception V3</u>***

```
import numpy as np
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input as
inception_preprocess_input
```

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt


train_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\training'
validation_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\validation'
test_dir = r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\
dataset\testing'

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    fill_mode='nearest'
)

def load_and_preprocess_image(file_path):
    img = load_img(file_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = inception_preprocess_input(img_array)
    return img_array


def load_data_from_directory(directory):
    data = []
    labels = []
    class_indices = {}
    for idx, class_name in enumerate(sorted(os.listdir(directory))):
        class_indices[class_name] = idx
        class_dir = os.path.join(directory, class_name)
        for filename in os.listdir(class_dir):
            if filename.lower().endswith((".jpg", ".jpeg", ".png")):  # Supporting multiple image
formats
                img_path = os.path.join(class_dir, filename)
                img_array = load_and_preprocess_image(img_path)
                data.append(img_array)
                labels.append(idx)
    return np.array(data), np.array(labels), class_indices


train_data, train_labels, class_indices = load_data_from_directory(train_dir)
validation_data, validation_labels, _ = load_data_from_directory(validation_dir)
test_data, test_labels, _ = load_data_from_directory(test_dir)

# Convert labels to categorical format
num_classes = len(class_indices)
train_labels = to_categorical(train_labels, num_classes)
validation_labels = to_categorical(validation_labels, num_classes)
test_labels = to_categorical(test_labels, num_classes)


base_model = InceptionV3(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])



# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),            loss='categorical_crossentropy',
metrics=['accuracy'])

for layer in base_model.layers:
    layer.trainable = False  # Freeze all layers

# Callbacks
```

```
checkpoint = ModelCheckpoint('best_model_GoogleNet.h5', monitor='val_loss',
save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=4, min_lr=1e-6)

# Train the model
batch_size = 32
history = model.fit(
    datagen.flow(train_data, train_labels, batch_size=batch_size),
    epochs=50,
    validation_data=(validation_data, validation_labels),
    callbacks=[checkpoint, early_stopping, reduce_lr]
)


# training and validation accuracy plot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig('accuracy_plot.png')
plt.close()


# training and validation loss plot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig('loss_plot.png')
plt.close()



results_folder = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Results'
if not os.path.exists(results_folder):
    os.makedirs(results_folder)
os.rename('accuracy_plot.png', os.path.join(results_folder, 'accuracy_plot.png'))
os.rename('loss_plot.png', os.path.join(results_folder, 'loss_plot.png'))
```

14.4 Codes used for Weight Centric Mechanism

```
import numpy as np
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import load_img, img_to_array


from tensorflow.keras.applications.inception_v3 import preprocess_input as
preprocess_input_inception
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input as
preprocess_input_mobilenet
from tensorflow.keras.applications.efficientnet import preprocess_input as preprocess_input_effi
from tensorflow.keras.applications.resnet import preprocess_input as preprocess_input_res


from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_curve,
auc, precision_recall_curve, average_precision_score, confusion_matrix
import matplotlib.pyplot as plt

# Define paths to the models you want to use for majority voting

model1 = load_model(r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Results\MobileNet_Results\best_model_MobileNet.h5'
)
model2 = load_model(r'C:\Users\Aditya\OneDrive -
vitap.ac.in\Desktop\vit\Capstone_Project\Results\GoogleNet_Results\best_model_GoogleNet.h5'
)
```

```
model3 = load_model(r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Results\EfficientNet_Results\best_model_EfficientNet.h5')
model4 = load_model(r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Results\ResNet_Results\best_model_ResNet.h5')


test_dir = r'C:\Users\Aditya\OneDrive - vitap.ac.in\Desktop\vit\Capstone_Project\Facial_Recognition_System_for_Hindu-_Gods-master\dataset\testing'

def load_and_preprocess_image_Mobile(file_path):
    img = load_img(file_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = preprocess_input_mobilenet(img_array)
    return img_array

def load_and_preprocess_image_GoogleNet(file_path):
    img = load_img(file_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = preprocess_input_inception(img_array)
    return img_array

def load_and_preprocess_image_Efficient(file_path):
    img = load_img(file_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = preprocess_input_effi(img_array)
    return img_array

def load_and_preprocess_image_Resnet(file_path):
    img = load_img(file_path, target_size=(224, 224))
    img_array = img_to_array(img)
    img_array = preprocess_input_res(img_array)
    return img_array



def load_data_from_directory(directory,m):
    data = []
    labels = []
    class_indices = {}
    for idx, class_name in enumerate(sorted(os.listdir(directory))):
        class_indices[class_name] = idx
        class_dir = os.path.join(directory, class_name)
        for filename in os.listdir(class_dir):
            if filename.lower().endswith((".jpg", ".jpeg", ".png")):
                img_path = os.path.join(class_dir, filename)
                if m == 1:
                    img_array = load_and_preprocess_image_Mobile(img_path)
                elif m == 2:
                    img_array = load_and_preprocess_image_GoogleNet(img_path)
                elif m == 3:
                    img_array = load_and_preprocess_image_Efficient(img_path)
                else:
                    img_array = load_and_preprocess_image_Resnet(img_path)

                data.append(img_array)
                labels.append(idx)
    return np.array(data), np.array(labels), class_indices



test_data, test_labels, class_indices = load_data_from_directory(test_dir,1)
num_classes = len(class_indices)
predictions = model1.predict(test_data)
predicted_classes1 = np.argmax(predictions, axis=1)
accuracy1 = accuracy_score(test_labels,predicted_classes1)
print("MobileNet_Predictions")
print(predicted_classes1)


test_data, test_labels, class_indices = load_data_from_directory(test_dir,2)
predictions = model2.predict(test_data)
predicted_classes2 = np.argmax(predictions, axis=1)
accuracy2 = accuracy_score(test_labels,predicted_classes2)
print("GoogleNet_Predictions")
print(predicted_classes2)

test_data, test_labels, class_indices = load_data_from_directory(test_dir,3)
predictions = model3.predict(test_data)
```

```
predicted_classes3 = np.argmax(predictions, axis=1)
accuracy3 = accuracy_score(test_labels,predicted_classes3)
print("EfficientNet_Predictions")
print(predicted_classes3)

test_data, test_labels, class_indices = load_data_from_directory(test_dir,4)
predictions = model4.predict(test_data)
predicted_classes4 = np.argmax(predictions, axis=1)
accuracy4 = accuracy_score(test_labels,predicted_classes4)
print("ResNet_Predictions")
print(predicted_classes4)


class_array = [0] * 10

accuracies = [accuracy1, accuracy2, accuracy3, accuracy4]
print(accuracies)

final_predictions = []
for i in range(len(predicted_classes1)):
    class_array = [0] * 10
    class_array[predicted_classes1[i]] += accuracy1
    class_array[predicted_classes2[i]] += accuracy2
    class_array[predicted_classes3[i]] += accuracy3
    class_array[predicted_classes4[i]] += accuracy4
    final_predictions.append(class_array.index(max(class_array)))



#final_predictions = np.column_stack(final_predictions)
print("Final Predictions after weight based voting")
print(final_predictions)

accuracy = accuracy_score(test_labels, final_predictions)
f1 = f1_score(test_labels, final_predictions, average='weighted')
precision = precision_score(test_labels, final_predictions, average='weighted')
recall = recall_score(test_labels, final_predictions, average='weighted')

print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)



# Plot the confusion matrix
import seaborn as sns
import pandas as pd
conf_matrix = confusion_matrix(test_labels, final_predictions)
print("Confusion Matrix:")
print(conf_matrix)

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="Blues", xticklabels=class_indices.keys(),
yticklabels=class_indices.keys())
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

15. REFERENCES

1. **https://rishihood.edu.in/artificial-intelligence-and-robotics-a-journey-through-ancient-indian-texts/ Accessed on 08th November 2023**
2. **Aggarwal., Sahoo., Bansal., Sarangi.: Image Classification using Deep Learning: A Comparative Study of VGG-16, InceptionV3 and EfficientNet B7 Models (July 2023)**
3. **https://github.com/Adinp1213/Mythic_Vision Created in September 2023**
4. **Spagnolo., Perri., Frustaci., Corsonello.: Connected Component Analysis for Traffic Sign Recognition Embedded Processing Systems (December 2018)**
5. **Armi., Ershad.: Texture Image Analysis and Texture Classification Methods - A Review (2019)**
6. **Lalaoui., Mohamadi.: A comparative study of Image Region-Based Segmentation Algorithms (July 2013)**
7. **Natarajan V., Kumar M., Tamizhazhagan V.: Text Region Detection and Recognition in Natural Scene Images Using MSER and Convolutional Neural Network (November 2020)**
8. **Chen., Li., Bai., Yang., Jiang., Miao.: Review of Image Classification Algorithms Based on Convolutional Neural Networks (November 2021)**
9. **Kovalev., Liauchuk., Kalinovsky.: A comparison of conventional and deep learning methods of image classification on a database of chest radiographs. (June 2017)**
10. **Sharma., Phonsa.: Image Classification Using CNN (April 2021)**
11. **Tulasi., Kumar Kalluri.: Deep Learning and Transfer Learning Approaches for Image Classification (February 2019)**
12. **Sandler., Howard., Zhu., Zhmoginov., Chen.: MobileNetV2: Inverted Residuals and Linear Bottlenecks (Revised March 2019)**
13. **Tan., Quoc V. Le: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (Revised September 2020)**
14. **He., Zhang., Ren., Sun.: Deep Residual Learning for Image Recognition (December 2015)**
15. **Szegedy., Vanhoucke., Loffe., Shlens., Wolna.: Rethinking the Inception Architecture for Computer Vision (Revised December 2015)**
16. **Tragoudaras., Stoikos., Fanaras. Tziouvaras.: Design Space Exploration of a Sparse MobileNetV2 Using High-Level Synthesis and Sparse Matrix Techniques on FPGAs (June 2022)**
17. **https://pytorch.org/hub/pytorch_vision_mobilenet_v2 (Accessed on 13 November 2023)**
18. **Gorji., Shahabi., Sharma., Tande.: Combining deep learning and fluorescence imaging to automatically identify fecal contamination on meat carcasses (February 2022)**
19. **Suvaditya Mukherjee: The Annotated ResNet-50 (August 2022)**

20. https://cloud.google.com/tpu/docs/inception-v3-advanced (Accessed on 13 November 13, 2023)

21. https://pytorch.org/hub/pytorch_vision_inception_v3/ (Accessed on 13 November 13, 2023)

22. Martin Ward Powers.: Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation.(January 2011)

23. https://www.sciencedirect.com/science/article/pii/S2352340922010642

24. https://app.creately.com/d/2HuVVoAPyoT/edit