Buddhist Painting Recognition System

Project Report

submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

Mr. Tashi Phuntsho

Mr. Prem Bdr Bhandari

Ms. Nar Maya Tamang

Under the Guidance of

Mr. Karma Wangchuk

(June, 2023)

INFORMATION TECHNOLOGY DEPARTMENT

COLLEGE OF SCIENCE AND TECHNOLOGY

ROYAL UNIVERSITY OF BHUTAN

ROYAL UNIVERSITY OF BHUTAN

COLLEGE OF SCIENCE AND TECHNOLOGY

INFORMATION TECHNOLOGY DEPARTMENT

CERTIFICATE

This is to certify that the B.E. a project titled **“Buddhist Painting Recognition System”,** which is being submitted by **Mr. Tashi Phuntsho (02190161), Mr. Prem Bdr Bhandari (02190151) and Ms. Nar Maya Tamang (02190143)** in partial fulfillment of the requirement for the award of the degree of Bachelor of Engineering in Information Technology is a record of students work carried out at College of Science and Technology, Phuentsholing under my supervision and guidance.

Mr. Karma Wangchuk

Project Guide

Acknowledgment

This project would not have been successful without the valuable guidance from our project guide, Mr. Karma Wangchuk. His unwavering support and helpful assistance played a crucial role in shaping the direction of our efforts, guiding us toward a victorious accomplishment, and his sagacious dedication illuminated our path to success. His profound encouragement has served as a guiding light, igniting our spirits and fueling our pursuit of excellence.

Moreover, we sincerely thank the Thimphu handicraft shop owners and other religious painting shop owners in and around Bhutan who graciously lent us their unwavering support in our pursuit of data collection. Their generosity in allowing us to gather images from their shop has played an indispensable role in the realization of our project's objectives. Without their collaboration, our endeavors would have been futile, and we remain forever grateful for their trust.

We also extend our profound gratitude to the Information and Technology Department, particularly to the esteemed review panel members. Their astute leadership and meticulous evaluation have been vital in assessing our progress, offering constructive criticism, and providing insightful comments that have significantly enriched our project.

Lastly, we again offer our deepest appreciation to Mr. Karma Wangchuk, a paragon of excellence and a distinguished faculty member of the ITD, as the final year project coordinator, his exceptional coordination and adept management have been instrumental in ensuring the success of the AS2022-SS2023 final year project.

Abstract

Computer vision is a multidisciplinary field that focuses on empowering computers with the capability to understand and interpret digital images and videos. Bhutan, as a country with a strong Buddhist presence, holds profound cultural and religious value in its Buddhist paintings. They often depict specific deities, religious figures, or symbolic representations of Buddhist teachings. However, as the number of Buddhist paintings is vast, identification and classification can be challenging for both tourists and locals. In this context, the application of computer vision technology becomes crucial for the preservation, analysis, and appreciation of Bhutan's rich artistic heritage. To address the challenge of identification and comprehension faced by tourists and locals, we aimed to develop a Buddhist painting recognition system utilizing advanced image processing techniques. The initial phase involved the meticulous creation of a comprehensive dataset, comprising 20 distinct classes of Buddhist paintings. This dataset formed the basis for training and comparing robust recognition models using state-of-the-art CNN and YOLOv5 architectures. The optimized model was subsequently integrated into web and mobile applications, enabling real-time detection. Through evaluation, the YOLOv5-based model achieved an impressive mean Average Precision (mAP) score of 82%, demonstrating its efficacy and accuracy.

Terminology

1. **Dataset**: Collection of data samples used for model training which is divided into training and validation.
2. **Convolutional Neural Network (CNN):** It is a deep learning algorithm specifically designed for image recognition and processing.
3. **You Only Look Once (YOLO):** An algorithm for real-time object detection that real timers an image into a grid and predicts class probabilities and bounding boxes.
4. **mean Average Precision(mAP):** widely used evaluation metric in object detection and information retrieval tasks.
5. **Epoch:** a complete cycle in the training process of a model.
6. **Batch size:** refers to the number of data samples processed together in a single iteration during training or inference in machine learning models.
7. **Hyperparameter:** Parameters set by the user before the training process to influence the performance of the model.
8. **Dropout:** neural network technique that lowers overfitting and boosts generalization by temporarily ignoring randomly chosen neurons during training.
9. **Batch Normalization:** a technique used in deep learning to normalize each layer’s inputs by subtracting the mean and dividing by the standard deviation of the batch, which helps stabilize and accelerate the training process.
10. **Confusion Matrix:** It is a tabular format that provides an organized perspective of the classification model's accuracy and errors by listing the number of true negative, true positive, false positive, and false negative predictions.
11. **Model:** A program or algorithm that is trained on a dataset to make predictions or to perform tasks.

List of Abbreviations

Sl.no	Terms	Descriptions
1	CNN	Convolutional Neural Network
2	YOLO	You Only Look Once
3	mAP	Mean Average Precision
4	ILSVRC	ImageNet Large Scale Visual Recognition Challenge
5	ReLU	Rectified Linear Unit
6	VGG	Visual Geometry Group
7	Adam	Adaptive Moment Estimation
8	GUI	Graphical User Interface
9	HTML	Hypertext Markup Language
10	CSS	Cascading Style Sheets
11	UI	User Interface
12	ITD	Information Technology Department
13	IBM	International Business Machines
14	CIFAR	Canadian Institute for Advanced Research
15	MNIST	Modified National Institute of Standards and Technology database
16	SGD	Stochastic Gradient Descent
17	CSPNet	Cross Stage Partial Network
18	PANet	Path Aggregation Network
19	JPEG	Joint Photographic Experts Group
20	PNG	Portable Network Graphics

List of Tables

List of Figures

Table of Content

CHAPTER 1: INTRODUCTION

1.1 Background

Buddhism is a prominent religion that has a rich cultural heritage in various countries around the world. In Bhutan, a small Himalayan kingdom, Buddhism is the dominant religion, and it has a significant influence on the art, culture, and traditions of the country. Bhutan has a long history of Buddhist art, which includes murals, thangkas, sculptures, and other forms of art that depict the teachings of Buddha and various Buddhist deities.

Advancement in deep learning and computer vision have made it possible to create intelligent systems that can recognize objects and images with high accuracy. The ability of these systems to automate image recognition tasks has made them useful in various fields, including healthcare, agriculture, and transportation. In this project, we aim to develop a deep learning-based system for recognizing Buddhist paintings and images, specifically those related to Bhutanese Buddhist art.

The recognition of Buddhist paintings is a difficult challenge due to the intricate details and variances found in the artwork. Moreover, there is no known dataset of Bhutanese Buddhist paintings and the application to recognize religious gods and deities. Therefore, the project aims to address these issues by collecting and annotating a dataset of Bhutanese Buddhist paintings and images and developing a deep learning-based recognition system using CNN (convolutional neural networks) and YOLO (You Only Look Once) object detection algorithms.

The project's objectives are to collect and annotate a dataset of Bhutanese Buddhist paintings and images, train a deep learning model, evaluate the system's performance, and develop a deep learning-based recognition system. The project will contribute to the preservation and promotion of Bhutanese Buddhist art by creating a tool that can be used to identify and classify various forms of Buddhist paintings. Additionally, it has the potential to be used in various fields such as tourism, art, and education.

1.2 Motivation

Bhutan is a country that is known for its unique cultural and religious heritage, and its people have a deep respect for the religious figures that are depicted in the paintings found in every household. However, recognizing these figures and understanding their significance can be a challenge for visitors to the country who come from different cultural backgrounds. This can lead to a missed opportunity to fully appreciate the rich cultural diversity of Bhutan and its traditions.

Furthermore, even Bhutanese people themselves may struggle to identify certain religious figures depicted in these paintings, especially younger generations who may not have been exposed to the same level of cultural education as their predecessors. This poses a threat to the preservation of Bhutan's religious heritage and the pride that comes with being Bhutanese. To address this problem, our project aims to develop a Buddhist painting recognition system using deep learning and image classification techniques. Despite the success of these techniques in various applications such as face recognition and object detection, there is currently no known dataset for Buddhist paintings and no existing application for recognizing these important religious figures in Bhutan.

By developing this system, we hope to contribute to the preservation of Bhutan's cultural heritage and enable visitors to appreciate the significance of these paintings. Moreover, our project also has the potential to inspire future research and development in the field of image recognition, particularly for applications that are specific to cultural and religious contexts.

1.3 Aim

To develop a Buddhist painting recognition system using image processing.

1.4 Objectives

- Conduct a literature review on Image processing.
- Select a suitable Deep learning Algorithm for developing Buddhist painting recognition system.
- Prepare a dataset on Buddhist paintings.
- Train a deep learning model.
- Design a graphical user interface.
- Deploy the model.

1.5 Scope

The scope of the project was to:

- Produce a Buddhist painting dataset:

  There was no well-known dataset available for the paintings of Buddhist gods and deities, so our goal was to identify the most common Buddhist gods and deities and collect a dataset consisting of 20 different classes with 2000 images per class. We performed the necessary preprocessing on the collected images.

- Train Buddhist painting recognition model:

  To train a deep learning model using Convolutional Neural Networks and YOLO.

- Design a Graphical User Interface for web and mobile applications:

  A graphical user interface was designed for the web and mobile application, and the model was integrated into it. The application was capable of detecting and naming Buddhist paintings, providing brief and useful information about them. The recognition process involved uploading images or using the camera for live detection.

CHAPTER 2: RELATED WORKS AND STUDIES

In this chapter, the related works for the project are thoroughly examined, with a particular emphasis on the deep learning algorithms employed. The discussed algorithms include the Convolutional Neural Network (CNN), AlexNet, LeNet-5, VGG-16, VGG-19, and the object detection algorithm known as YOLO (You Only Look Once). By exploring these related works, a solid foundation is laid for the project, providing insights into the strengths and limitations of each algorithm and guiding the subsequent development and implementation process.

2.1 Deep Learning

A branch of machine learning that focuses on teaching artificial neural networks how to analyze and extrapolate information from massive volumes of data. It consists of neural networks of multiple layers which helps in optimizing the accuracies of the predictions (*What Is Deep Learning? | IBM*, n.d.).

2.2 Convolutional Neural Network (CNN)

Deep learning algorithms that is frequently employed for image categorization problems. It draws inspiration from how the human brain processes visual data (Bhatt et al., 2021). Convolutional, pooling, and fully linked layers are among the many layers that make up CNNs. A series of filters are applied by the convolutional layer in order to extract features from the input image. The feature maps are subsequently down sampled by the pooling layer to minimize the input size. The input image is classified by the fully connected layer using the retrieved characteristics.

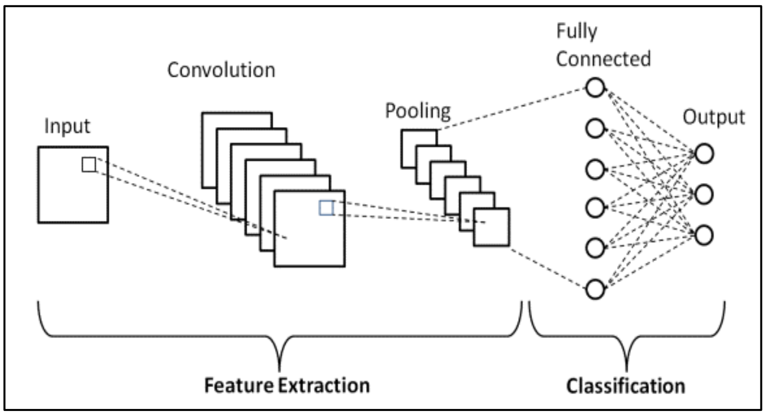


Figure 1: Basic CNN architecture (Gurucharan, 2022)

CNNs have the benefit of being able to automatically extract significant features from the input image without the requirement for manual feature extraction. Due to this, they are particularly beneficial for tasks such as image classification, face recognition, and object recognition. CNNs have shown state-of-the-art performance on a wide range of image classification tasks and have been widely used in many applications.

The convolutional neural network was used for image recognition on the MNIST dataset and CIFAR-10 dataset. MNIST has 70,000 images (28 x 28 pixels in size) for 10 classes labeled from (0-9) and CIFAR-10 has 60,000 color images (32 x 32 in size) for 10 classes. The CNN model used for the MNIST dataset had three convolution layers and a fully connected layer and five convolution layers and a dense layer for the CIFAR-10 dataset. Dropout and data augmentation were used to solve the overfitting problem and fine-tuned some hyperparameters. The final accuracy they got was 99.6% for MNIST data and 80.17% for CIFAR-10 data (Chauhan et al., 2018).

2.2 AlexNet

Alex Krizhevsky and his team created the convolutional neural network architecture known as AlexNet in 2012. The first deep learning architecture to considerably outperform conventional machine learning algorithms on the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) was created for this challenge (Alzubaidi et al., 2021).

Five convolutional layers make up AlexNet, which are followed by three fully connected layers. The network is given nonlinearity by applying a ReLU (rectified linear unit) activation function. After the first two convolutional layers, there is a subsequent max pooling layer that reduces the image's spatial size by half. Following this, the next three convolutional layers are succeeded by another max pooling layer that reduces the spatial size by a factor of three. After the fully connected layers, a SoftMax layer is added to create class probabilities.

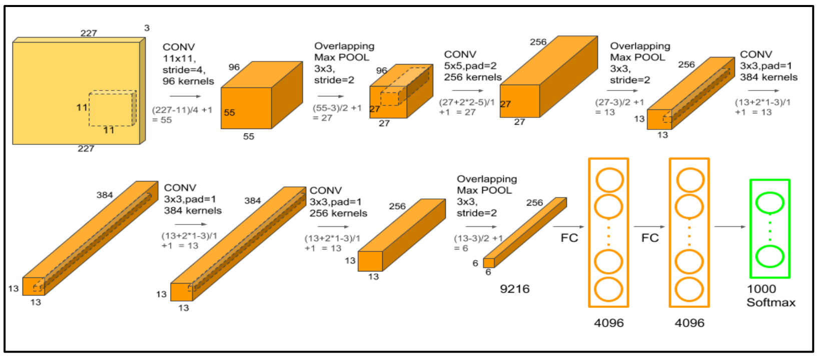

CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
9216
FC
4096
FC
4096
1000 Softmax

Figure 2: AlexNet in Depth (Musstafa, 2022)

One of the main innovations of AlexNet is the use of data augmentation techniques, to artificially expand the size of the training set and prevent overfitting. Another innovation is the use of dropout regularization, which randomly removes neurons during training to prevent co-adaptation (Krizhevsky et al., 2012).

AlexNet achieved one of the best results on the ILSVRC 2012 challenge, with a top-5 error rate of 15.3%. It paved the path for the development of deeper and more complex CNN architectures, to achieve even better performance.

2.3 LeNet-5

LeNet-5 is a pioneering convolutional neural network (CNN) architecture developed by Yann LeCun in the 1998, aimed at recognizing handwritten digits. It consists of multiple convolutional and pooling layers, followed by fully connected layers, and played a crucial role in the advancement of deep learning and computer vision.

A 5x5 convolutional layer with 6 filters makes up the first layer. The size of the output from the first layer is reduced by a subsampling layer in the second layer. Another convolutional layer of size 5x5 with 16 filters makes up the third layer. Another subsampling layer makes up the fourth layer. The fully connected layers are the fifth and sixth. A SoftMax layer with the total outputs makes up the last layer.

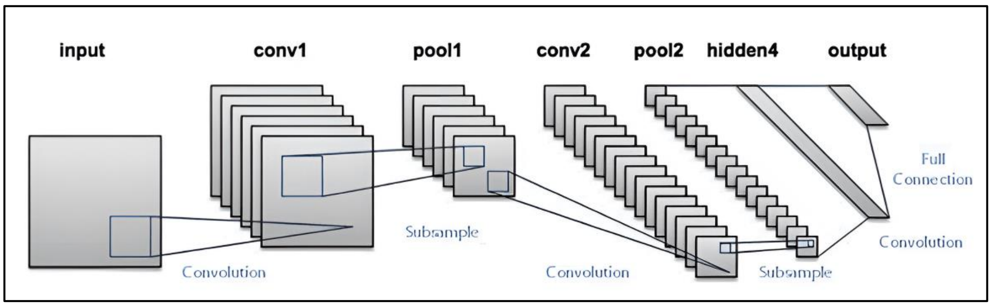


Figure 3: LeNet Architecture (Rosebrock, 2023)

LeNet-5 was a breakthrough in its time and achieved state-of-the-art results on handwritten digit recognition tasks. It introduced the idea of utilizing convolutional neural networks for image classification tasks and served as an inspiration for numerous following models, including the widely used AlexNet.

2.4 VGG-16

The Visual Geometry Group at Oxford University developed the convolutional neural network architecture known as VGG16. It is frequently employed in tasks involving picture recognition and categorization. "VGG" in VGG16 stands for "Visual Geometry Group," and "16" denotes the number of layers (16). VGG16's architecture consists of 3 fully connected layers and 13 convolutional layers. There are two different kinds of convolutional layers: 2D convolutional layers and 2D max pooling layers. Small receptive fields of 3x3 filters are used in the convolutional layers because they are thought to be a suitable balance between accuracy and efficiency.

The input image is of dimensions 224x224x3, where the first two dimensions correspond to the height and width of the image, and the third dimension corresponds to the RGB color channels. The output is a set of probabilities for each of the image classes that the network has been trained on.

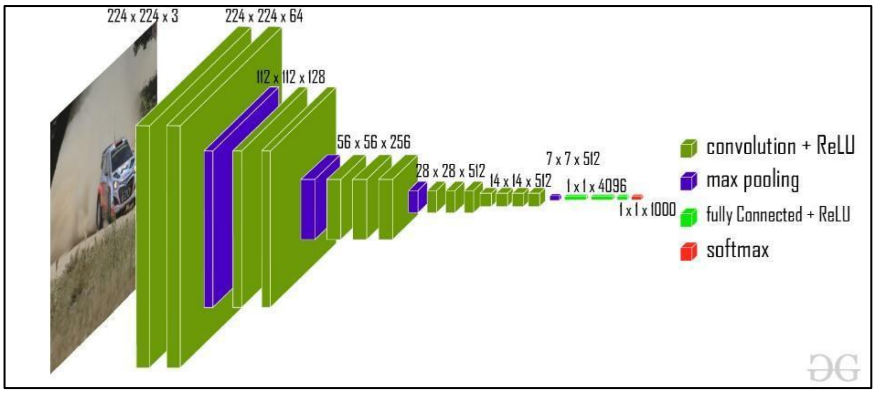


Figure 4: VGG-16 architecture (GeeksforGeeks, 2023)

VGG16 has been trained on the ImageNet dataset, where the network was trained to classify images into one of 1,000 different categories, such as "dog," "cat," or "car." It achieved state-of-the-art results with an error rate of 7.3% on the ImageNet dataset. VGG16 is known for its simplicity and ease of implementation, as well as its high accuracy.

2.5 VGG-19

VGG-19 is a deep convolutional neural network introduced in 2014 by the Visual Geometry Group at Oxford University. It achieved state-of-the-art performance on image recognition tasks and is widely used in computer vision research. It is an extension of the VGG16 architecture and contains 19 layers.

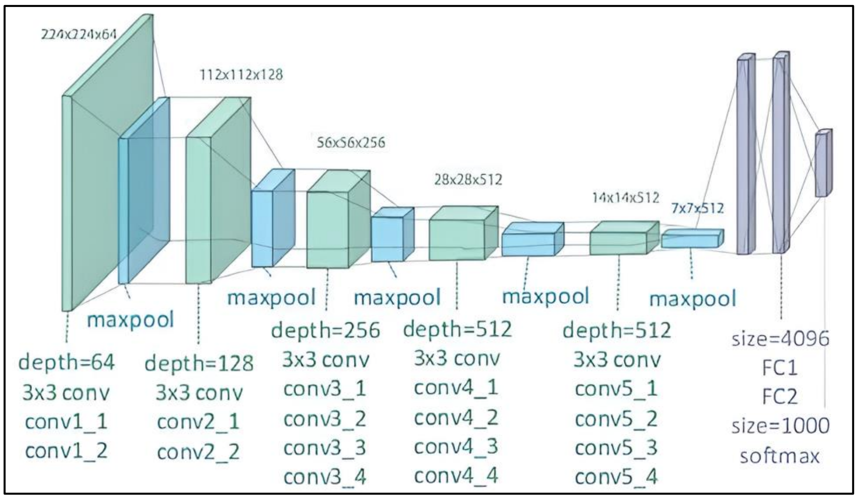


Figure 5: VGG-19 architecture (Zheng et al., 2018)

The architecture has a similar structure to VGG16, but with additional convolutional layers that allow for more complex features to be learned. Edges and corners are learned in the first few layers of VGG19, while more complicated characteristics like shapes and textures are learned in the deeper layers, and the fully connected layer is used for classification purposes.

One of the main advantages of the VGG19 architecture is its simplicity and uniformity, with all layers having the same output size and using the same filter size. This simplicity makes it easy to understand and implement, while still achieving high accuracy on various image classification tasks. VGG19 has demonstrated exceptional performance on multiple image classification benchmarks achieving better results like that of ILSVRC in 2014.

2.6 Optimization Technique

Optimization techniques are used to improve model accuracy by minimizing loss, adjusting weights and biases, and tuning hyperparameters. These techniques, such as stochastic gradient descent (SGD) and its variants, iteratively update the model weights based on the gradients of the loss function. They help navigate the high-dimensional parameter space to converge toward an optimal solution.

Adaptive Moment Estimation (Adam) optimizer is a popular stochastic gradient descent optimization algorithm that uses adaptive learning rates and momentum to efficiently update network weights and biases, improving training speed and accuracy. In their seminal paper, Kingma and Ba (2015) introduced Adam as an optimization method that estimates first and second moments of gradients to compute adaptive learning rates for each parameter. The algorithm has gained significant attention due to its ability to converge faster and handle different types of gradients effectively. Several studies have demonstrated the effectiveness of Adam in various deep learning tasks, such as image classification and many more. These findings highlight the importance of optimization techniques like Adam in enhancing the performance and convergence of deep learning models.

2.7 Activation Function

Activation functions are crucial components of deep learning models as they add nonlinearity, enabling the network to recognize complex patterns and make predictions. They transform the input of a neuron into its output and determine the activation level of the neuron. Activation functions play a key role in capturing both local and global nonlinear relationships within the data, enabling the network to model complex functions and improve the representation power of neural networks (Glorot et al., 2011). By applying activation functions at each layer, deep learning models can learn hierarchical representations and extract higher-level features.

The Rectified Linear Unit (ReLU) was introduced by Nair and Hinton (2010), ReLU replaces negative inputs with zero, while leaving positive inputs unchanged. ReLU offers several advantages over traditional activation functions, such as computational efficiency and the ability to resolve the vanishing gradient problem. The simplicity and effectiveness of ReLU have contributed to its widespread adoption, and it has been shown to improve the performance of deep neural networks in various tasks, including image recognition (Krizhevsky et al., 2012) and natural language processing (Maas et al., 2013).

The output layer uses the Softmax function. Softmax converts previous layer outputs to probabilities over multiple classes, aiding classification by producing a probability distribution, enabling the interpretation of the network's outputs as class probabilities. It normalizes the output values and ensures that the probabilities sum up to one. The Softmax function is instrumental in multi-class classification tasks, allowing the network to assign a probability to each class and make informed decisions based on these probabilities. The SoftMax function enables probabilistic inference and has greater classification accuracy (Kuleshov & Ermon, 2018; Mikolov et al., 2010). Its ability to generate class probabilities makes it valuable not only for classification but also for tasks such as uncertainty estimation and ensemble modeling.

2.8 YOLO

You Only Look Once, popularly referred to as YOLO (object detection algorithm) revolutionized real-time object detection in computer vision tasks. Proposed by Redmon et al. (2016), YOLO takes a different approach by performing object detection directly on a single neural network, eliminating the need for region proposal techniques used in traditional methods. YOLO creates grids from input images predicting bounding boxes and the respective class for each grid cell, enabling efficient and precise object detection. It achieves impressive speed while maintaining competitive accuracy, making it well-suited for real-time applications (Redmon et al., 2016).

Since its inception, YOLO has undergone several improvements and versions. Following the original YOLO, subsequent versions were introduced to enhance performance and address limitations. YOLOv2, proposed by Redmon and Farhadi (2017), incorporated features like anchor boxes and multi-scale training to improve localization accuracy. YOLOv3, presented by Redmon and Farhadi (2018), introduced additional improvements such as feature extraction at multiple scales and a prediction method that handles objects of different sizes. These iterations further improved detection accuracy and expanded the range of detectable object sizes.

A notable advancement in the YOLO series is YOLOv4, introduced by Bochkovskiy et al. (2020). YOLOv4 utilized advanced techniques such as spatial pyramid pooling, backbone network (CSPDarknet53), and PANet for feature fusion.

More recently, YOLOv5, developed by Glenn Jocher as an independent project, gained significant attention in the computer vision community. YOLOv5 focused on improving performance and ease of use with a streamlined architecture, incorporating novel techniques such as the EfficientDet-D7 backbone and AutoML for hyperparameter optimization. YOLOv5 achieved impressive results in object detection benchmarks, combining accuracy, speed, and simplicity, making it popular among researchers and practitioners in the field.

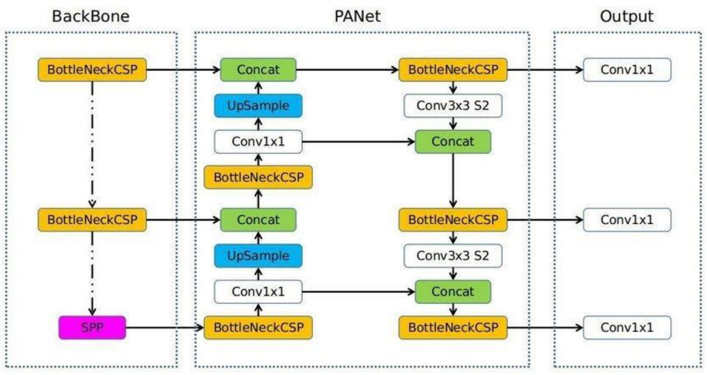


Figure 6: Architecture of YOLOv5 (Imane, 2022)

The YOLOv5 architecture consists of a backbone network, a neck for feature fusion, and a detection head for generating bounding box predictions and class probabilities. It utilizes anchor boxes for accurate bounding box adjustments and employs focal loss for improved handling of challenging objects. This architecture enables YOLOv5 to efficiently and accurately detect objects in real-time (Jocher, 2020).

CHAPTER 3: METHODOLOGY

In this chapter, a comprehensive overview will be presented, outlining the key stages encompassed in the development of the system. The discussion will encompass data collection, preprocessing techniques, model training methodologies, and the subsequent integration of the trained models into the system. By examining each stage in detail, readers will gain a clear understanding of the systematic approach adopted throughout the creation process.

3.1 Project Overview

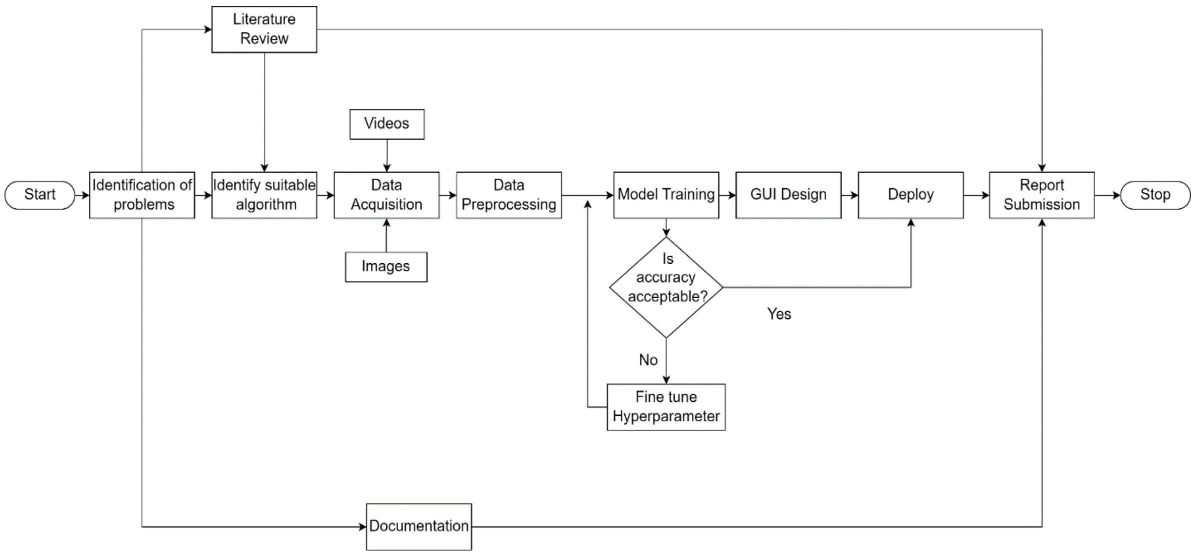
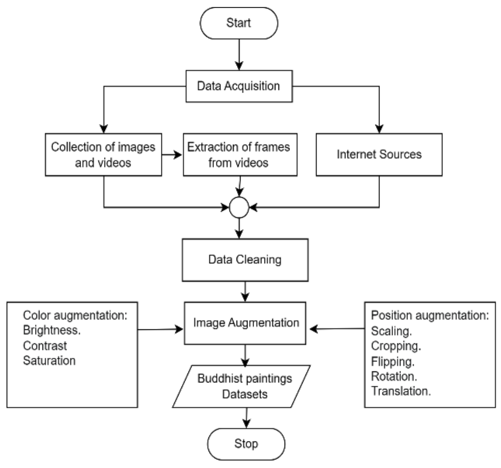


Figure 7: Project Overview

We started by identifying the issues and selecting a project topic. Then, we conducted a thorough literature review and documented our progress throughout the project. After studying numerous related works, we chose an appropriate algorithm for our topic. The next step was to prepare the dataset by acquiring and pre-processing the data. We trained our model using VGG16, a CNN-based architecture, and YOLO. Meanwhile, we also designed the graphical user interface for our application. Once the model training was completed, we integrated the VGG16 model into the web application and the YOLO model into the mobile application.

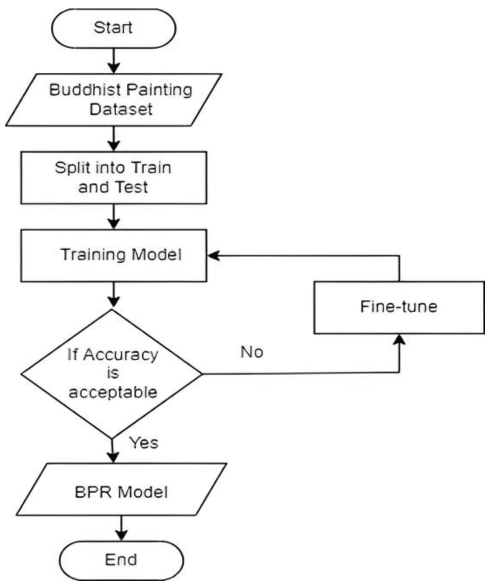
The first step in the dataset preparation process was data acquisition, which involved gathering information in the form of images and videos from various sources and extracting the frames from the videos. The undesirable and corrupted photos were then removed through data cleaning. After data cleansing, image augmentation was used to add variety to the dataset and boost its size.



Start
Data Acquisition
Collection of images and videos
Extraction of frames from videos
Internet Sources
Data Cleaning
Color augmentation: Brightness. Contrast Saturation
Image Augmentation
Position augmentation: Scaling. Cropping. Flipping. Rotation. Translation.
Buddhist paintings Datasets
Stop

Figure 8: Dataset Preparation

Once the dataset is prepared, it is divided into training and validation sets. Then the model is trained, and if the obtained accuracy is not satisfactory, the model is fine-tuned and retrained. This iterative process continues until the best or optimal result is achieved.



Start
Buddhist Painting Dataset
Split into Train and Test
Training Model
Fine-tune
If Accuracy is acceptable
No
Yes
BPR Model
End

Figure 9: Model training process

For the application development, the GUI was initially prototyped using a prototyping tool. Then, the application was developed based on the finalized GUI prototype. Finally, the model was integrated into the application, allowing it to seamlessly utilize the functionalities and predictions of the model.

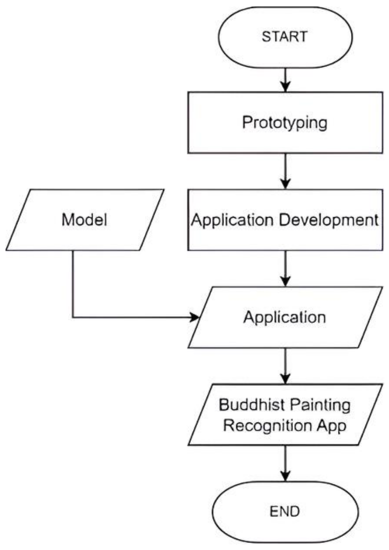


Figure 10: Web and Mobile application development process.

3.2 Working of the system.

Web application

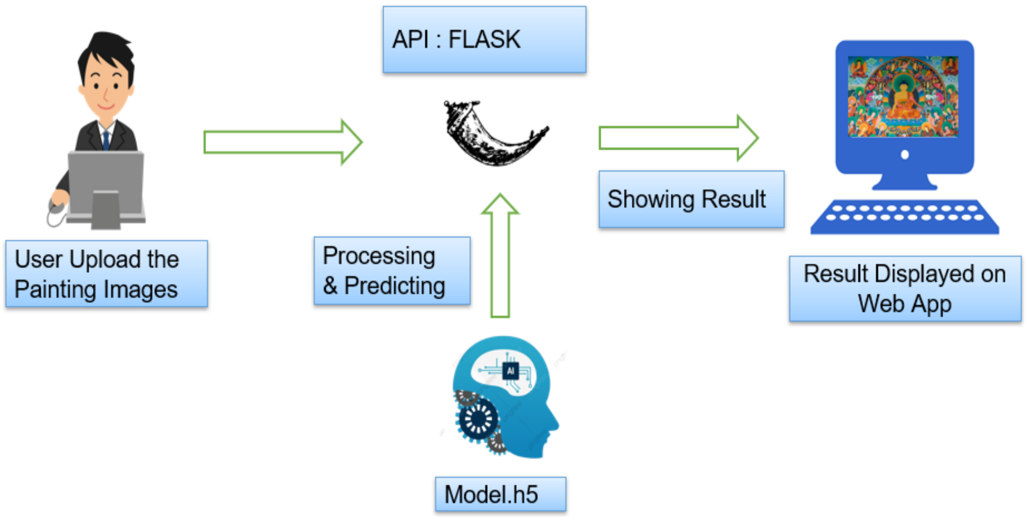
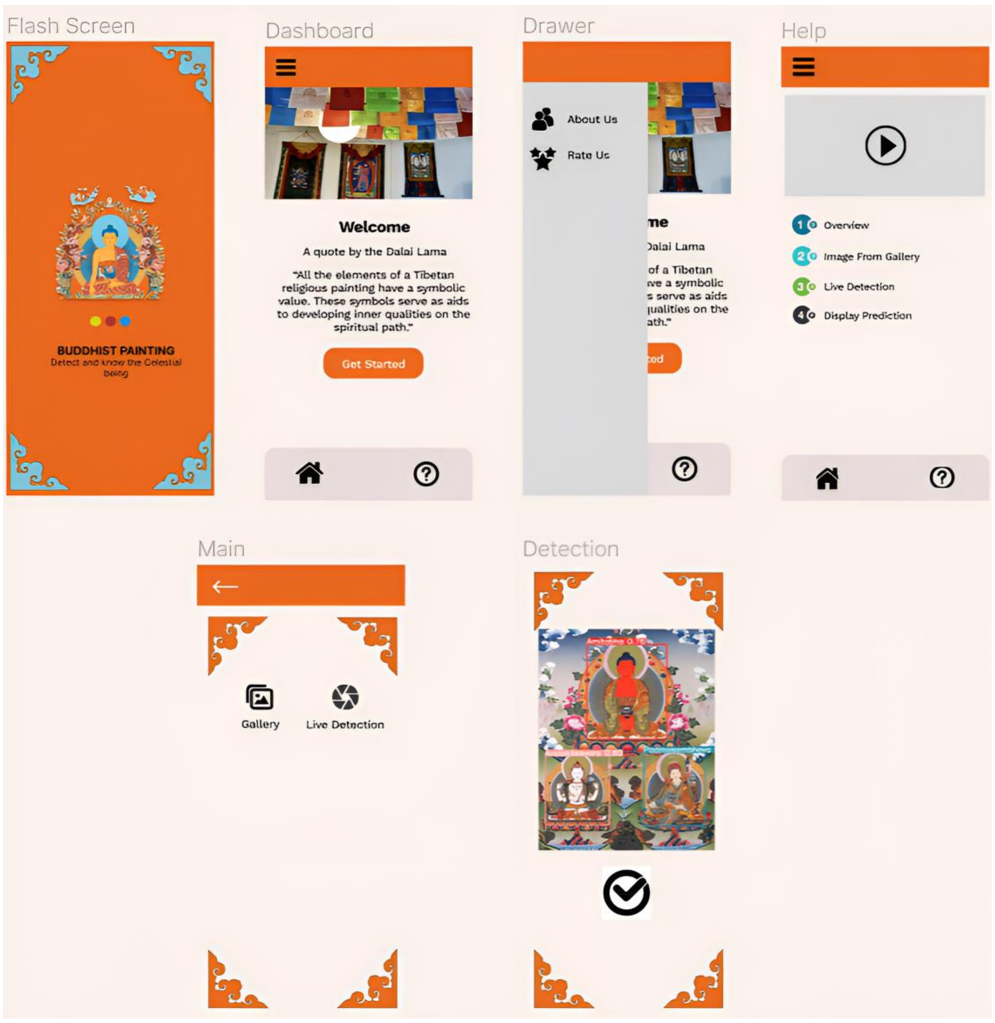


Figure 11: Working of Web Application

In our Flask-based application, users can input a Buddhist painting, and our trained model will classify it. The painting will undergo analysis and classification, utilizing our robust model. The classification result will be promptly displayed, providing users with valuable insights into the artwork and its artistic characteristics. Experience seamless classification in our application.

Mobile Application

Prototype (Figma)



"All the elements of a Tibetan religious painting have a symbolic value. These symbols serve as aids to developing inner qualities on the spiritual path."

Figure 12: Figma prototype for mobile application

Upon launching the application, users will be welcomed to a home page offering convenient sections such as Help, Settings, and About Us. The main detection page will provide two options: uploading an image or performing live detection. Powered by our trained YOLO object detection model, users can opt for either method. Uploading an image will trigger the model to analyze and detect objects, while live detection will utilize the device's camera to instantly identify objects in real-time. On selecting either option, users will be directed to a details page where comprehensive information about the detected objects will be displayed.

3.3 Environmental setup

3.3.1 CNN

Identification of suitable algorithm

To select a suitable model, we first selected CNN-based architectures after going through several related works and then trained those pre-trained base models for the dataset of 5 classes. Then select the best one to carry on the training for the complete dataset.

Table 1: Number of Epochs and Batch Size for Model Selection

Models	**Epochs**	**Batch size**
LeNet-5	20	64
AlexNet	20	64
VGG-16	20	64
VGG-19	20	64
CNN	20	64

Model training

The following specifications were used for the selected CNN model training:

Table 2: Hyperparameter used for selected model training

Epoch	**Batch size**	**Dropout**
100	32	0.2
100	32	0.3
100	64	0.2
100	64	0.3

To overcome the limitations of CNN, which is designed for single image classification rather than real-time detection, we incorporated YOLO (You Only Look Once) into our training process. YOLO enables real-time object detection by analyzing an entire image in one pass. By leveraging the power of YOLO, our mobile application can efficiently detect objects and provide accurate results in real-time, enhancing the user experience and expanding the possibilities of visual analysis.

3.3.2 YOLO

Identification of suitable algorithm

A version of the YOLO series, YOLOv5, provides several persuasive arguments for its adoption above earlier iterations. Both accuracy and speed have significantly increased in YOLOv5. It presents a simplified design that balances model complexity with inference speed, making it extremely effective for real-time object detection tasks.

The YOLOv5s model, in particular, stands out for its quick inference capabilities and lightweight design. YOLOv5s enables effective deployment on resource-constrained devices without compromising detection performance thanks to a smaller model size and lower computational demand. The remarkable results obtained by YOLOv5, along with the benefits of YOLOv5s in terms of speed and efficiency, make it an appealing option for a variety of real-time object identification applications (Jocher, 2020).

3.4 Model training

For the yolo model training, the following hyperparameters were used for the pretrained model:

Table 3: Number of Epoch and Batch size for training YOLO model

Epoch	**Batch size**
100	32
100	64

CHAPTER 4: DATASET PREPARATION

4.1 Introduction

This chapter briefs on the process of Buddhist painting dataset preparation. The dataset was collected for the twenty commonly found Buddhist deities, bodhisattvas and the Buddhas. The dataset was prepared separately for CNN and YOLO implementation since CNN focuses on classifying an image and YOLO on detecting multiple objects in an image.

4.2 Dataset Preparation for CNN

4.2.1 Data collection

The dataset prepared for the project was gathered from both videos and images, mostly from the regions of Thimphu and Phuentsholing. The images and videos were taken with a focus on a single class image, considering the angles of view and lighting effects. The length of the videos ranged from 25 to 30 seconds, and two video frames were extracted per second.

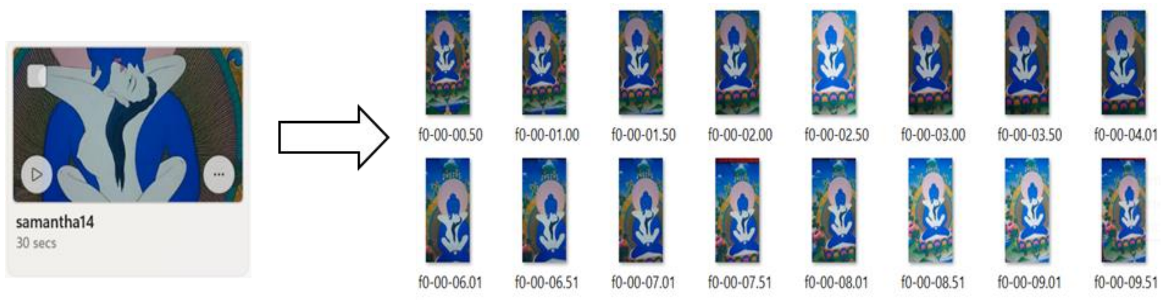


Figure 13: Frame extraction from video

For the 20 classes, the number of video samples ranged mostly from 25 to 30. To expand the dataset, additional data was gathered through web crawling. In total, 40,000 images were collected for the 20 classes, whereby each class contained 2,000 images.

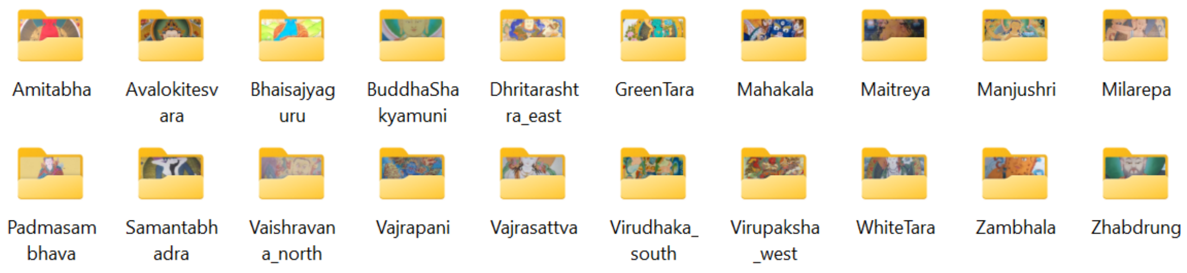


Figure 14: 20 classes Dataset folders

4.2.2 Data cleaning

After the extraction and collection of images, unwanted and duplicate images were removed, and the images were converted into the same formats, either jpg or png, which are supported for CNN training. To address computational limitations, we compressed the massive 19 GB dataset, resizing it to 227 x 227 dimensions. By doing so, we significantly reduced the dataset's total size to a more manageable 350 MB, ensuring efficient processing without compromising the necessary data for analysis.

4.2.3 Data Augmentation

Data augmentation is a powerful technique that increases the size and diversity of a dataset, making it easier to collect and more effective for training models. It can improve model generalization and prevent overfitting by providing more varied examples for training. (Hernández-García et al., 2019). By applying various transformation methods, the dataset is augmented, increasing its size and diversifying the data for model training. For the dataset, about 500 augmented images were generated for each class whereby 1500 images were the original images. The augmented data was created by using augmentation techniques such as rotation, horizontal flipping, adding noise, varying brightness, saturation, and blurring and sharpening the image.

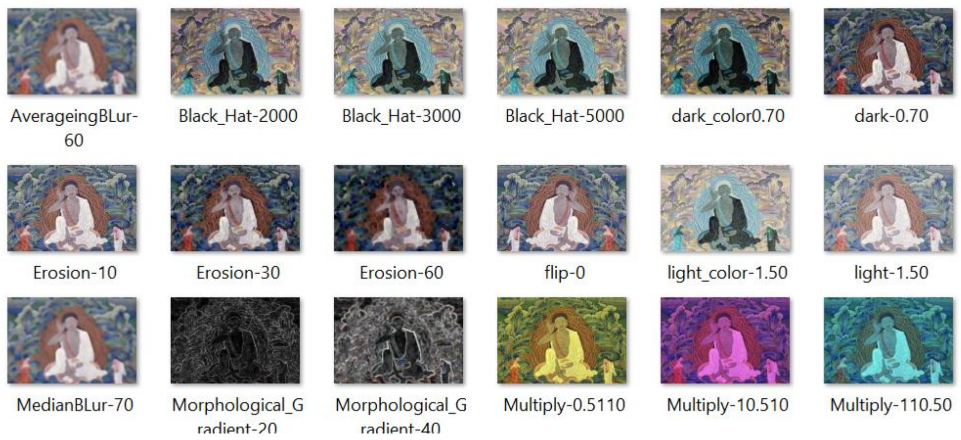


Figure 15: Example of image augmentation

4.2.4 Data split

The entire dataset was split into two parts, i.e., training and validation. As per the findings of Prashanth et al. (2020), the ratio of 80:20 gave better accuracy, so we also split our dataset 80:20 ratio, 80 percent for training and 20 percent for validation.

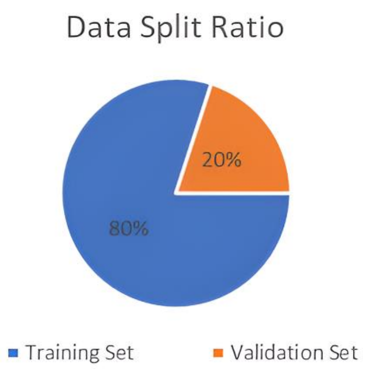


Figure 16: Dataset split ratio

4.3 Dataset preparation for YOLO

4.3.1 Data collection

The process of collecting data for YOLO training was similar to that of CNN, where images and videos were collected and two frames per second were extracted. Additionally, images containing multiple classes were also included in the dataset. Overall, 15,000 images were collected for YOLO training.

Figure 17: Yolo Data collection videos.

3.3.2 Data annotation

Data annotation is the process of labeling or tagging data with metadata that provides context and meaning. In the object detection process, data annotation is crucial for creating accurate and effective models. We used an online tool called makesense.ai for the annotation process. With makesense.ai, we were able to create bounding boxes around the objects in the images and assign them to specific classes.

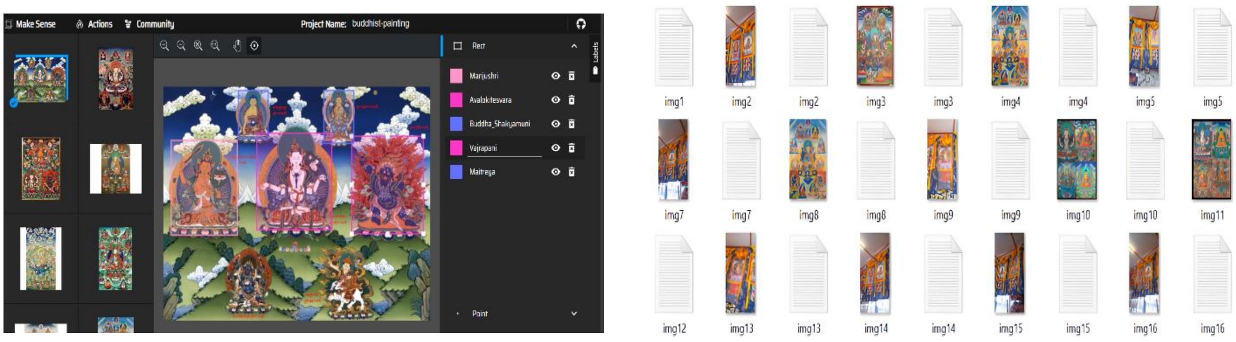


Figure 18: Data Annotation

3.3.3 Data split

After the completion of data annotation, the images and its corresponding labels were equally divided into 80:20 ratio of dataset for train and validation.

Table 4: Dataset for Buddhist Painting Detection system

Sl.no	Class	Image per class (CNN)	Annotations per class (YOLO)
1	Amitabha	1500	1080
2	Avalokiteshvara/Chenrezig	1500	1152
3	Bhaisajyaguru/Medicine Buddha	1500	1024
4	Buddha Shakyamuni	1500	1783
5	Vajrapani/Chana Dorji	1500	1101
6	Green Tara	1500	1129
7	Mahakala	1500	1089
8	Maitreya	1500	1691
9	Manjushri/Jampelyang	1500	1228
10	Milarepa	1500	1053
11	Padmasambhava/Guru Rinpoche	1500	1570

12	Samantabhadra /Kuntuzangpo	1500	1248
13	Vaishravana_North	1500	724
14	Vajrasattva/Dorji Sempa	1500	664
15	White Tara	1500	754
16	Zambhala	1500	752
17	Zhabdrung Ngawang Namgyel	1500	1003
18	Virudhaka_North	1500	825
19	Dhritarashtra_East	1500	804
20	Virupaksha_West	1500	882

CHAPTER 5: RESULT AND DISCUSSION

5.1 Introduction

CNN (Convolutional Neural Networks) brought about significant transformation in the domain of computer vision, enabling significant advancements in image classification tasks. Buddhist Painting image classification is a challenging task due to the complexity and diversity of artistic styles. CNNs have proven to be highly effective in the painting classification domain, enabling automated recognition and categorization of paintings. By training CNN models on large-scale painting datasets, it becomes possible to classify paintings.

CNNs can capture intricate details and nuances in paintings, including brush strokes, color palettes, and composition. They can learn to distinguish between different artistic styles and identify the characteristic elements of renowned painters. This has significant implications for art historians, museum curators, and art enthusiasts, as it allows for automated analysis, organization, and retrieval of vast art collections.

Moreover, CNNs can be used in various applications related to painting images, such as forgery detection, style transfer, and artistic image generation. By leveraging the power of CNNs, researchers and artists can explore new dimensions in the world of art and visual creativity.

In this chapter, we will delve into the application of CNNs in Buddhist painting image classification, specifically focusing on the LeNet-5, AlexNet, VGG-16, VGG-19, and a custom CNN model. We compare their performance and select the best performing model with identical hyperparameters, investigate the fine-tuning of the selected model, and demonstrate the integration of the selected model into a web application. The insights gained from this chapter provide valuable knowledge for advancing computer vision techniques in the field of art analysis and appreciation.

5.2 CNN Based model selection with same hyperparameters

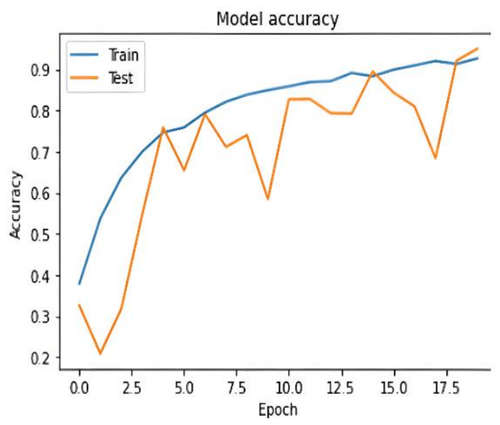
CNN Model Selection is a crucial step in designing effective convolutional neural network architectures for specific computer vision tasks. In this section, we consider several renowned CNN models and evaluate their performance in Buddhist painting image classification. Firstly, we examine the LeNet-5 architecture proposed by LeCun et al. (1998). Second, we investigate the AlexNet architecture introduced by Krizhevsky et al.

(2012), which was pivotal in popularizing deep learning. AlexNet incorporates multiple convolutional layers with a large number of parameters to capture complex image features. Next, we explore the VGG-16 architecture presented by Simonyan and Zisserman (2014), characterized by its deep structure with 16 weight layers. VGG-16 is known for its uniformity and achieved impressive results in various image recognition challenges. Additionally, we consider the VGG-19 architecture, an extension of VGG-16 with 19 weight layers, to compare its performance. Furthermore, we introduce a custom CNN architecture tailored for painting image classification. Finally, to ensure fair comparison and selection, we carefully tune the hyperparameters for each model with learning rate of 0.1, epochs of 20, batch size of 64, and optimization algorithm with Adam, while keeping them consistent across the models. This systematic evaluation of different CNN architectures and hyperparameter settings enables us to identify the most suitable model for Buddhist painting image classification.

Table 5: CNN model comparison results

Model	**Epochs**	**Batch Size**	**Time taken**	**Training**		**Validation**	
				Accuracy (%)	Loss	Accuracy (%)	Loss
LeNet-5 Architecture	20	64	1 hr 22 minutes	63.69	0.9442	73.10	0.7337
AlexNet Architecture	20	64	1 hr 49 minutes	68.51	0.1544	73.25	1.3303
VGG-16 Architecture	**20**	**64**	**2 hr 11 minutes**	**92.65**	**0.3162**	**95.00**	**0.1816**
VGG-19 Architecture	20	64	2 hr 10 minutes	89.64	0.2816	88.55	0.3197
CNN Architecture	20	64	1 hr 51 minutes	85.15	0.4249	78.90	0.7882

The table 5 illustrate the evaluation of performance of LeNet-5, AlexNet, VGG-16, VGG-19, and a custom CNN, the VGG-16 architecture emerged as the best-performing model for painting image classification. With its deep structure and uniformity, VGG-16 demonstrated superior accuracy and robustness. Therefore, VGG-16 was selected as the optimal model from the above-mentioned models based on the evaluation results.

a) 

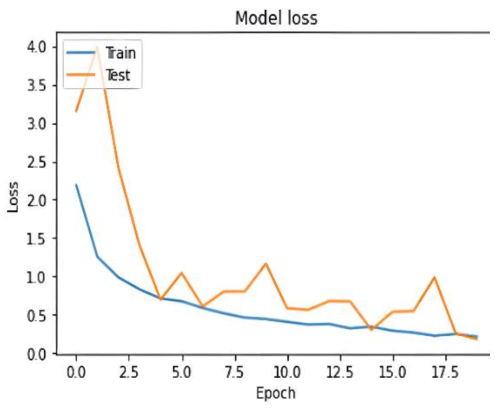
b) 

Figure 19: VGG-16 Model: a) Accuracy vs Epoch and b) Loss vs Epoch

Figure 19, presents two metrics of the selected VGG-16 model. a) It illustrates the accuracy vs epoch, showing the improvement in accuracy over time. b) It displays the loss vs epoch, indicating the reduction in loss during the training process. This graph offers valuable insights into the model's performance, demonstrating the positive trends of increasing accuracy and decreasing loss over time.

5.3 Selected Model fine tuning

VGG-16, the selected model for Buddhist painting image classification, underwent fine-tuning to optimize its performance. Fine-tuning involves adjusting the hyperparameters. The hyperparameters modified included the learning rate, batch size, batch normalization, dropout and epochs.

During fine-tuning, we have used batch normalization, dropout, and the learning rate was carefully tuned with *1e-3 * 0.9 ** x* to ensure an appropriate balance between rapid convergence and avoiding overfitting. A lower learning rate was chosen to allow the model to converge gradually and prevent drastic changes in weights. The batch size was adjusted to optimize memory usage and computational efficiency while maintaining a sufficient number of samples for stable training.

Regarding the optimization algorithm, different options were explored to identify the most effective approach. In this case, the Adam optimizer was chosen for its adaptive learning rate mechanism and momentum optimization, which helped in achieving faster convergence and better generalization.

The accuracy of the VGG-16 model significantly improved, the model demonstrated enhanced capability in differentiating artistic styles, recognizing Buddhist paintings and the fine-tuned VGG-16 model exhibited improved performance in terms of classification accuracy, robustness, and generalization.

Table 6: Results for the fine-tuned model

Epochs	**Batch Size**	**dropout**	**Training**		**Validation**	
			Accuracy (%)	Loss	Accuracy (%)	Loss
100	32	0.2	85.15	0.4436	79.80	0.6755
100	32	0.3	90.56	0.3141	85.43	0.3659
100	**64**	**0.2**	**91.53**	**0.2546**	**94.13**	**0.2121**

Table 6 illustrates the evaluation outcomes of the VGG-16 model's performance by experimenting with various hyperparameters, including dropout and batch size. Among the configurations, the highest accuracy of 91.53% was achieved when utilizing a batch size of 64 and a dropout rate of 0.2. Due to overfitting, the model struggled to accurately generalize the classification of Buddhist paintings, resulting in some misclassifications, this can be improved by increasing the dataset and augmentation.

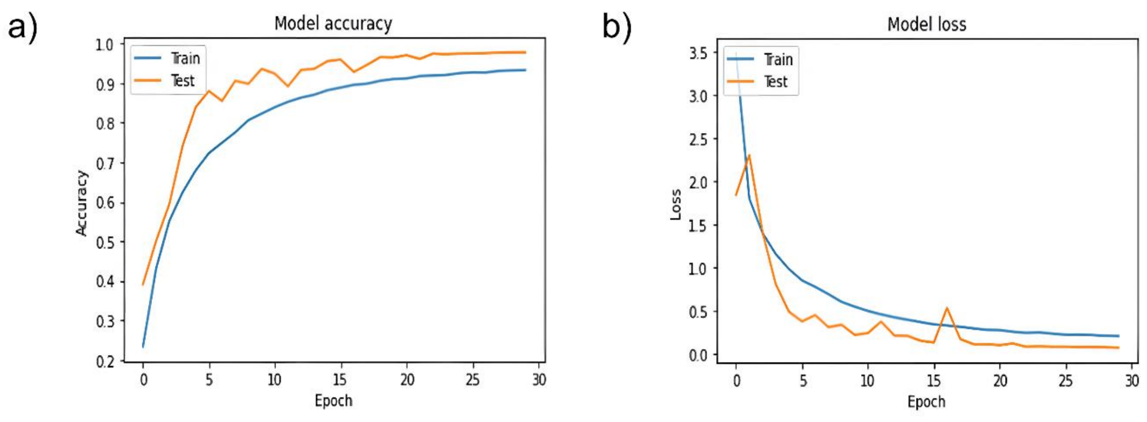
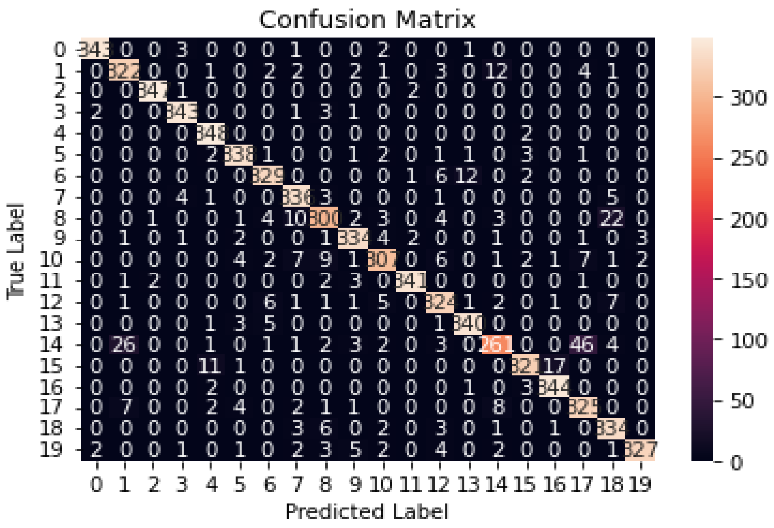


Figure 20: Fine-tuned VGG-16 model: a) Accuracy vs Epoch and b) Loss vs Epoch

Figure 20 In Figure 20, the selected model's accuracy and loss are graphically displayed, providing a visual representation of its performance. This information can be used to make informed decisions about the model's effectiveness and potential for improvement. The graph provides insights into the model's accuracy improvement and loss reduction.

```
1.    Amitabha
2.    Avalokitesvara
3.    Bhaisajyaguru
4.    BuddhaShakyamuni
5.    Dhritashtra_East
6.    GreenTara
7.    Mahakala
8.    Maitreya
9.    Manjushri
10.   Milerapa
11.   Padmasambhava
12.   Samantabhadra
13.   VaishravanaNorth
14.   Vajrapani
15.   Vajrasattva
16.   Virudhaka_South
17.   Virupaksha_West
18.   WhiteTara
19.   Zambhala
20.   Zhabdrung_Ngawang_Namgyel
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	343	0	0	3	0	0	0	1	0	0	2	0	0	1	0	0	0	0	0	0
1	0	322	0	0	1	0	2	2	0	2	1	0	3	0	12	0	0	4	1	0
2	0	0	347	1	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
3	2	0	0	343	0	0	0	1	3	1	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	348	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
5	0	0	0	0	2	338	1	0	0	1	2	0	1	1	0	3	0	1	0	0
6	0	0	0	0	0	0	329	0	0	0	0	1	6	12	0	2	0	0	0	0
7	0	0	0	4	1	0	0	336	3	0	0	0	1	0	0	0	0	0	5	0
8	0	0	1	0	0	1	4	10	300	2	3	0	4	0	3	0	0	0	22	0
9	0	1	0	1	0	2	0	0	1	334	4	2	0	0	1	0	0	1	0	3
10	0	0	0	0	0	4	2	7	9	1	307	0	6	0	1	2	1	7	1	2
11	0	1	2	0	0	0	0	0	2	3	0	341	0	0	0	0	0	1	0	0
12	0	1	0	0	0	0	6	1	1	1	5	0	324	1	2	0	1	0	7	0
13	0	0	0	0	1	3	5	0	0	0	0	0	1	340	0	0	0	0	0	0
14	0	26	0	0	1	0	1	1	2	3	2	0	3	0	261	0	0	46	4	0
15	0	0	0	0	11	1	0	0	0	0	0	0	0	0	0	321	17	0	0	0
16	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	3	344	0	0	0
17	0	7	0	0	2	4	0	2	1	1	0	0	0	0	8	0	0	325	0	0
18	0	0	0	0	0	0	0	3	6	0	2	0	3	0	1	0	1	0	334	0
19	2	0	0	1	0	1	0	2	3	5	2	0	4	0	2	0	0	0	1	327

Figure 21: Confusion matrix for Fine-tuned VGG-16 model

In Figure 21, the confusion matrix reveals that class 14-vajrapani was frequently misclassified as 17-virupaksha_west, occurring 46 times. This misclassification could be attributed to the similarity in visual patterns between the two classes. To mitigate this issue, augmenting the dataset or increasing its size can be done to introduce more variations, enabling the model to better distinguish between these classes.

5.4 Development of Flask Web Application

The Flask web framework provides a lightweight and flexible platform for building web applications. It is well-suited for integrating machine learning models into web environments. Flask offers features such as routing, templating, and request handling, making it ideal for creating a user-friendly interface to interact with the model.

The application employs HTML templates and CSS styling to create a visually appealing user interface. The Flask routing mechanism is utilized to handle requests from the client-side and process the uploaded images.

The trained VGG-16 model is deployed within the Flask web application. The model is loaded, and the necessary pre-processing steps are applied to the uploaded images to prepare them for classification. Flask's request handling capabilities allow the application to receive the uploaded images and pass them to the model for prediction. The classification results are obtained from the model and displayed to the user on the web interface.

The Flask web application, integrated with the VGG-16 model, undergoes thorough testing and evaluation. Various test cases are designed to validate the functionality and performance of the application. This includes uploading different Buddhist painting images, verifying the accuracy of the classification results, and assessing the responsiveness and user experience of the interface.

User Workflow

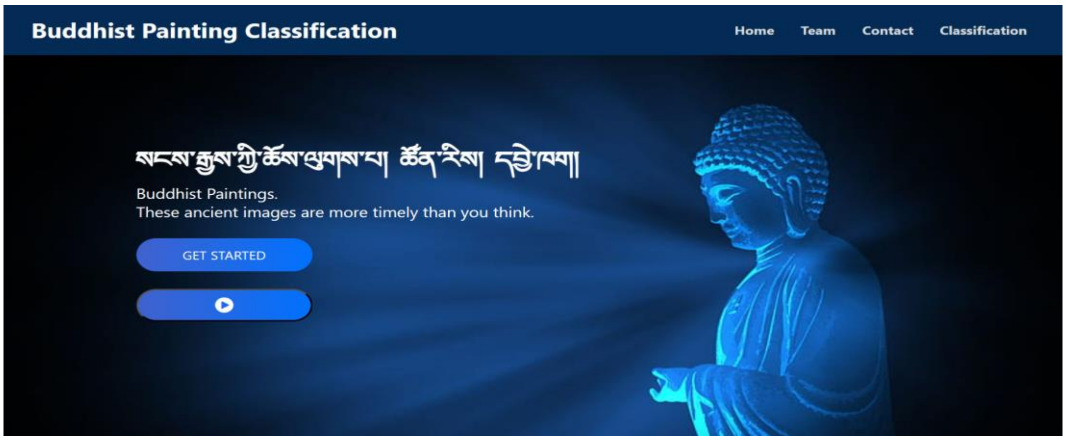
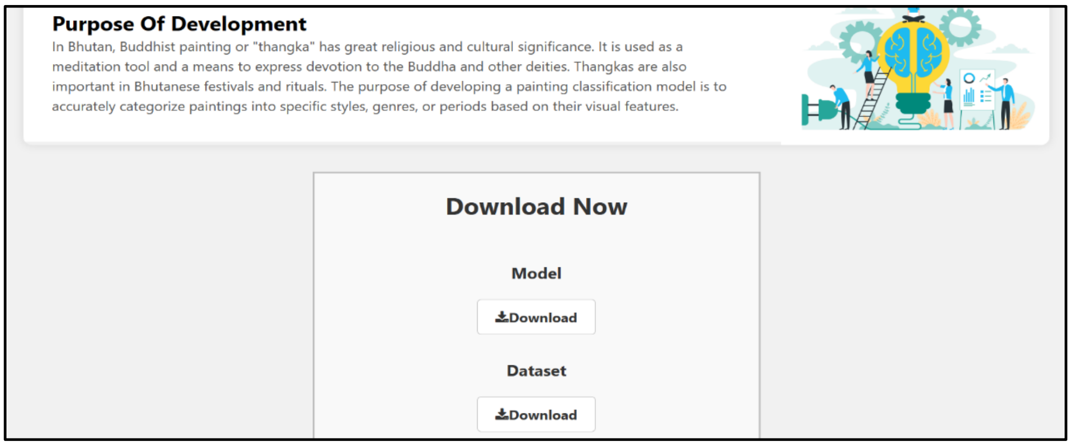

Buddhist Painting Classification
Home
Team
Contact
Classification
Buddhist Paintings.
These ancient images are more timely than you think.
GET STARTED

Figure 22: Web application Home page

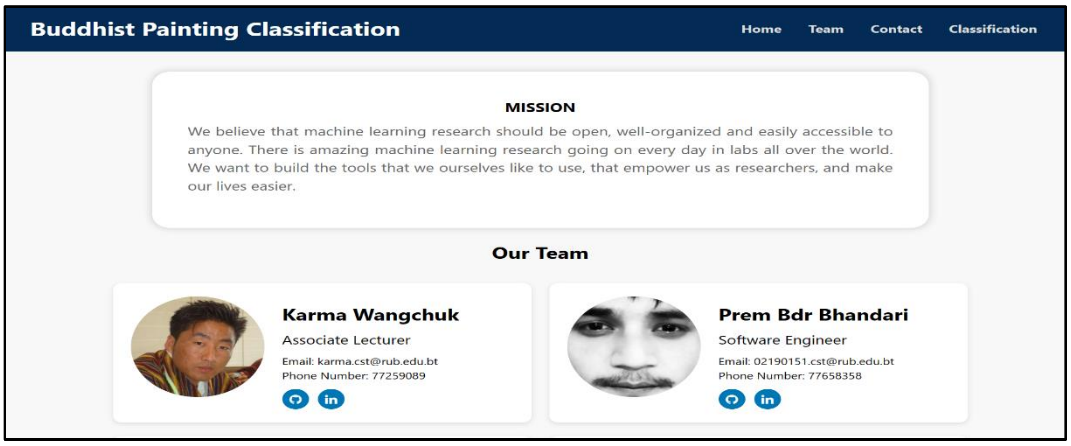
Model and dataset Download feature


Purpose Of Development

In Bhutan, Buddhist painting or "thangka" has great religious and cultural significance. It is used as a meditation tool and a means to express devotion to the Buddha and other deities. Thangkas are also important in Bhutanese festivals and rituals. The purpose of developing a painting classification model is to accurately categorize paintings into specific styles, genres, or periods based on their visual features.

Figure 23: Web application Model and Dataset download

Team Page


Buddhist Painting Classification

Home Team Contact Classification

MISSION

We believe that machine learning research should be open, well-organized and easily accessible to anyone. There is amazing machine learning research going on every day in labs all over the world. We want to build the tools that we ourselves like to use, that empower us as researchers, and make our lives easier.

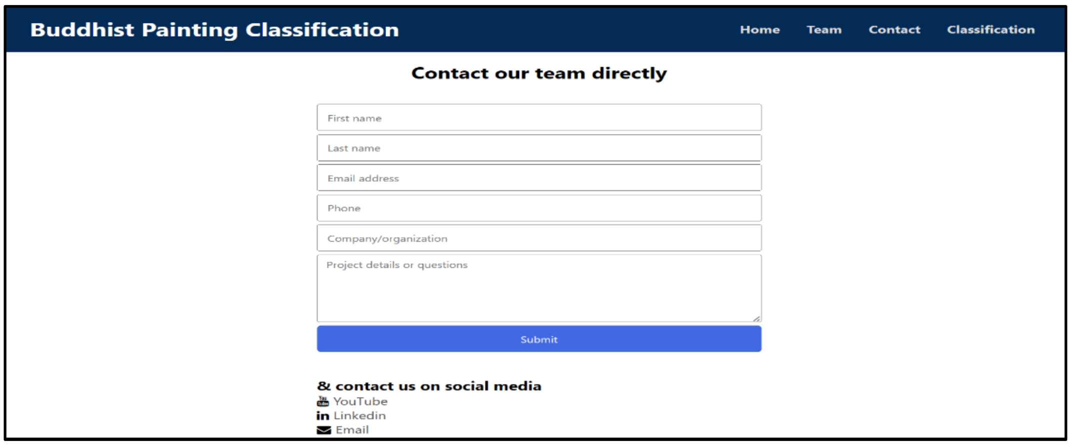
Our Team

Karma Wangchuk
Associate Lecturer
Email: karma.cst@rub.edu.bt
Phone Number: 77259089

Prem Bdr Bhandari
Software Engineer
Email: 02190151.cst@rub.edu.bt
Phone Number: 77658358

Figure 24: Web application Team page

Contact us page


Buddhist Painting Classification

Home Team Contact Classification

Contact our team directly

First name
Last name
Email address
Phone
Company/organization
Project details or questions

Submit

& contact us on social media
YouTube
Linkedin
Email

Figure 25: Web application Contact us page

Classification Page

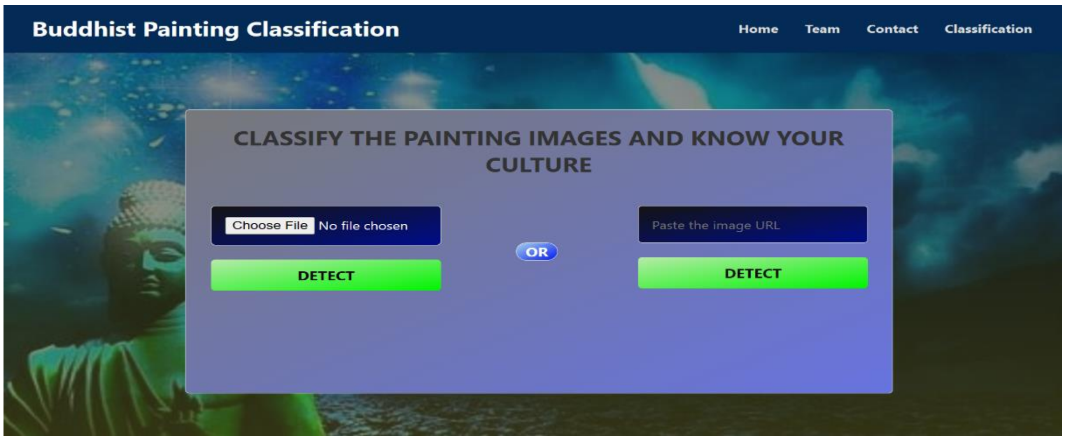

Buddhist Painting Classification
Home
Team
Contact
Classification
CLASSIFY THE PAINTING IMAGES AND KNOW YOUR CULTURE
Choose File No file chosen
DETECT
OR
Paste the image URL
DETECT

Figure 26: Web application Classification page

Process for actual Classification

To classify an image, simply select the image you want to classify. The application will then perform the classification process and display the results on the next page. Effortlessly analyze and categorize your images with just a few clicks.

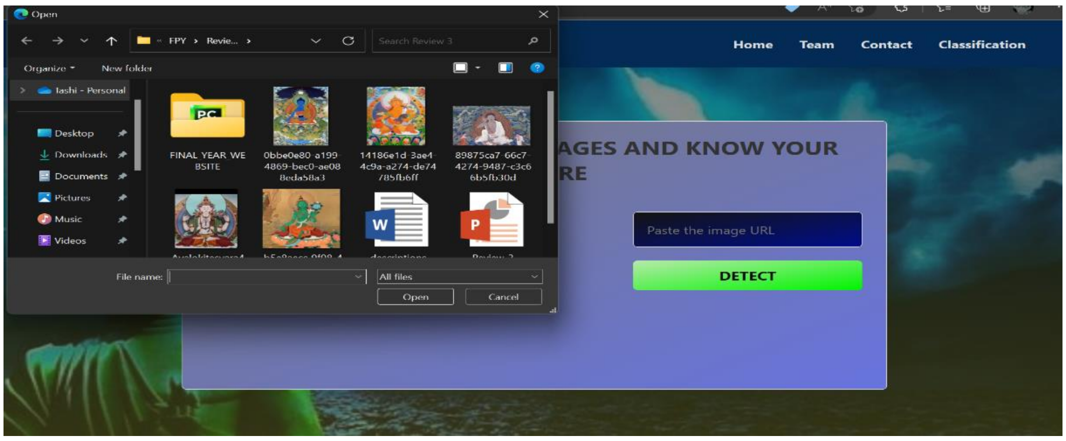


Figure 27:Web application Image uploading process

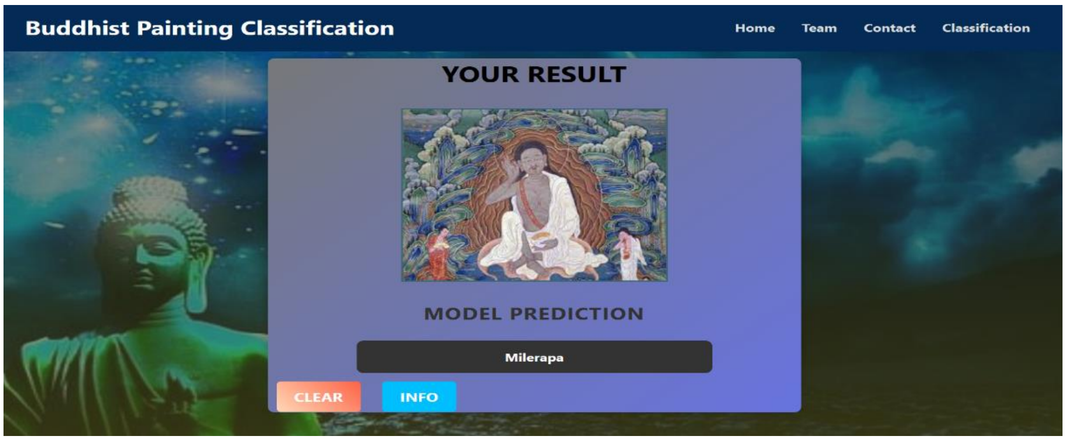


Figure 28: Result after classification

5.6 Yolo

In this research study, our objective was to train YOLOv5s (object detection model), using two distinct approaches. Firstly, we performed training from scratch, initializing the model with random weights and exclusively utilizing our custom dataset of Buddhist paintings. Secondly, we employed a transfer learning technique by fine-tuning the pretrained YOLOv5s model using our dataset. The subsequent section of this paper showcases the comprehensive results obtained from these experiments and offers an in-depth discussion, analyzing the performance and efficacy of both training methodologies. By exploring these approaches, we gained valuable insights into the applicability and effectiveness of YOLOv5s for detecting Buddhist paintings.

Table 7: Results of YOLO model training

Sl.no	**Model**	**Epoch**	**Batch Size**	**mAP**
1	Pretrained	100	32	82.0
2	Training from Scratch	100	32	78.0

Based on the information provided in table 7, it is apparent that training the model from a pretrained state yielded a higher mean Average Precision (mAP) of 82.0%. Therefore, this model variation has been chosen for integration into our mobile application.

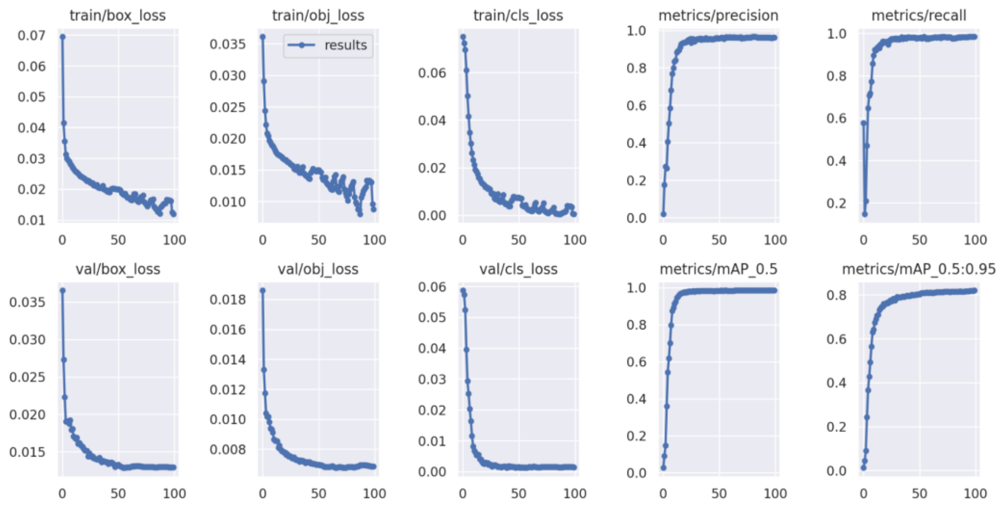


Figure 29:YOLO model Metrics graph

Figure 29 showcases the metrics graph of the YOLO model, focusing on the mean Average Precision (mAP) metric throughout the training process. This graph provides an overview of the model's performance and its ability to accurately detect objects, emphasizing the mAP as a measure of detection quality.

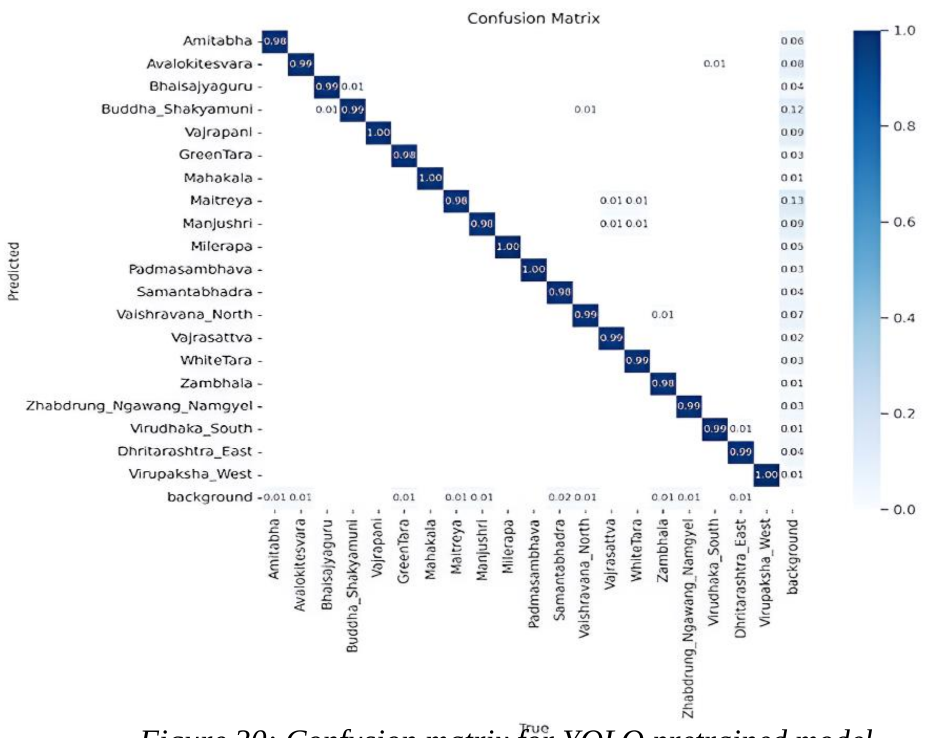


Figure 30: Confusion matrix for YOLO pretrained model

5.7 Mobile integration

Utilizing the flexibility of the Flutter framework, we seamlessly integrated the YOLO object detection model into our mobile application. This integration empowers our app to leverage the advanced object detection capabilities offered by YOLO, enabling accurate and efficient detection of objects in real-time. With YOLO integrated into our mobile application, we can provide users with enhanced functionalities and an improved experience.

Features:

1. Live Detection: Our app detects Buddhist paintings in real time, providing instant descriptions and insights with just a tap.
2. Upload from Gallery: Users can easily upload images of paintings from their gallery and access detailed descriptions and information.
3. Settings: Customize your app experience by choosing your preferred theme and personalizing the interface to match your style.
4. Help Section: Get step-by-step instructions and guidance on how to navigate and utilize all the features of our application.
5. About Us Section: Learn about the talented developers behind the app and discover their passion for creating innovative experiences for users like you.

5.7.1 Flutter

Flutter is an open-source UI toolkit developed by Google that enables developers to buid beautiful and natively compiled applications for mobile, web, and desktop platforms from a single codebase. Here are some key points about Flutter:

1. Cross-Platform Development: Flutter allows developers to write code once and deploy it across various platforms. It supports Android, iOS, web, and desktop platforms, allowing efficient and consistent development across different devices.
2. Fast and Reactive UI: Flutter utilizes a reactive framework that enables developers to create stunning and fluid user interfaces. It uses a powerful rendering engine that provides high-performance graphics and animations, resulting in smooth and responsive app experiences.
3. Hot Reload: Flutter's hot reload feature enables developers to instantly view code changes as they make them, improving development speed and efficiency.

5.7.2 Model integration challenges

Tflite plugin (Failed)

During the integration of the YOLOv5s model with a Flutter mobile application using the Tflite plugin, we encountered the following challenges that resulted in an unsuccessful integration:

- Version Compatibility: Ensuring compatibility between the version of the Tflite plugin, YOLOv5s model, and Flutter framework was difficult. The mismatch in versions caused conflicts and errors, preventing successful integration.
- Null Safety Issue for Camera Plugin: The integration with the camera plugin, which is crucial for live detection, presented challenges due to null safety issues. Proper handling of nullability was required to avoid crashes and ensure smooth functionality.
- Incompatible Score Type: The Tflite plugin expected float32 scores, while the YOLOv5s model provided int8 and float16 scores. This inconsistency in score types resulted in inaccurate object detection and affected the reliability of the integration.

flutter_tflite plugin (Failed)

During the integration of the yolov5s model with a Flutter mobile application using the flutter_tflite plugin, we encountered various challenges that ultimately led to an unsuccessful integration. Although we were able to successfully build the application, we faced a setback when we realized that the live detection functionality we desired relied on the Flutter_Tflite_helper package, which was discontinued. This limitation prevented us from achieving our goal of seamlessly integrating the yolov5s model and utilizing live detection within the Flutter mobile application. To overcome this obstacle, we explored alternative solutions for the desired functionality.

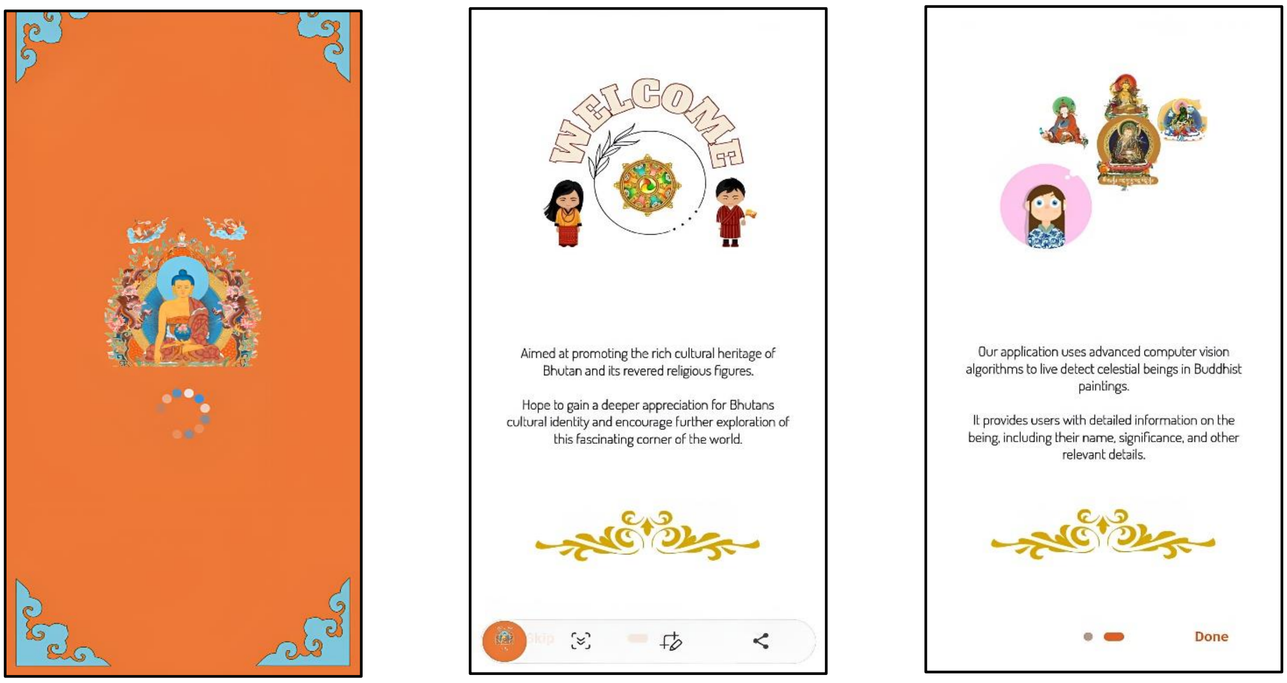
Flutter_pytorch plugin (Success)

After undergoing constant debugging and utilizing the Tflite format, the integration process faced setbacks and encountered failures. However, we persisted in our efforts and eventually discovered the Flutter_pytorch plugin, which employed the torchscript format. Transitioning to this new plugin proved to be a successful breakthrough. The adoption of the torchscript format provided enhanced compatibility and functionality, enabling us to effectively harness the capabilities of PyTorch within the Flutter mobile application. With the new plugin in place, we successfully overcame the limitations that were encountered during the previous approach. This achievement not only validated our decision but also opened up new possibilities for leveraging PyTorch models within the Flutter framework. Through perseverance and adaptability, we were able to identify a more suitable solution that aligned with our project requirements and led to a successful integration outcome.

5.7.3 User Workflow

Splash Screen and onboarding page:

We added a Bhutanese touch to the design, creating an onboarding page that provides a concise overview of the application's purpose and features.



Aimed at promoting the rich cultural heritage of Bhutan and its revered religious figures.

Hope to gain a deeper appreciation for Bhutans cultural identity and encourage further exploration of this fascinating corner of the world.

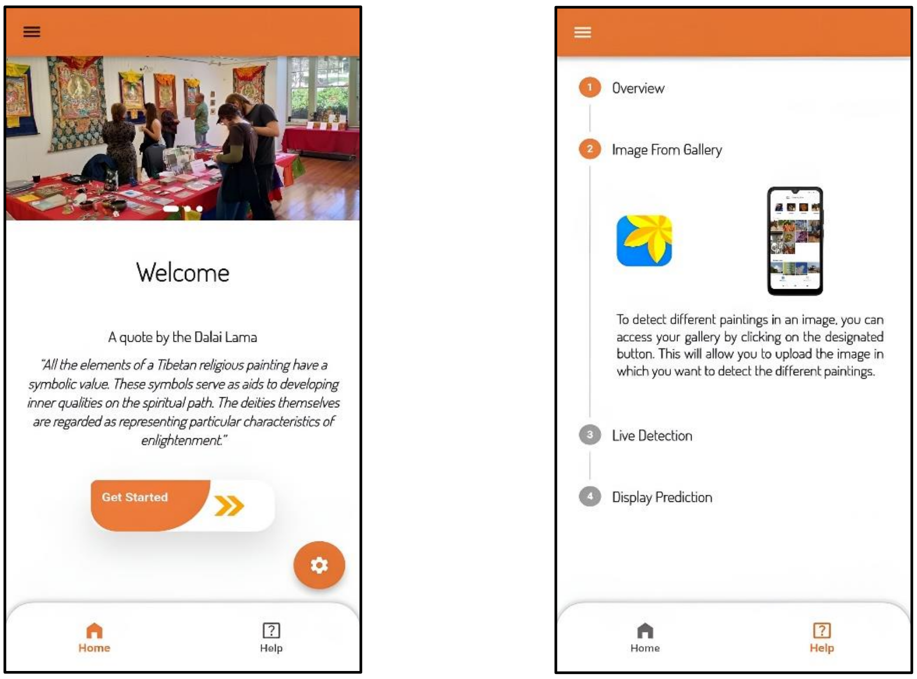
Skip

Our application uses advanced computer vision algorithms to live detect celestial beings in Buddhist paintings.

It provides users with detailed information on the being, including their name, significance, and other relevant details.

Done

Figure 31: Mobile Splash screen and onboarding page



Welcome

A quote by the Dalai Lama

"All the elements of a Tibetan religious painting have a symbolic value. These symbols serve as aids to developing inner qualities on the spiritual path. The deities themselves are regarded as representing particular characteristics of enlightenment."

Get Started

Home Help

1 Overview

2 Image From Gallery

To detect different paintings in an image, you can access your gallery by clicking on the designated button. This will allow you to upload the image in which you want to detect the different paintings.

3 Live Detection

4 Display Prediction

Home Help

Figure 32: Mobile Home page and the Help section

Setting:

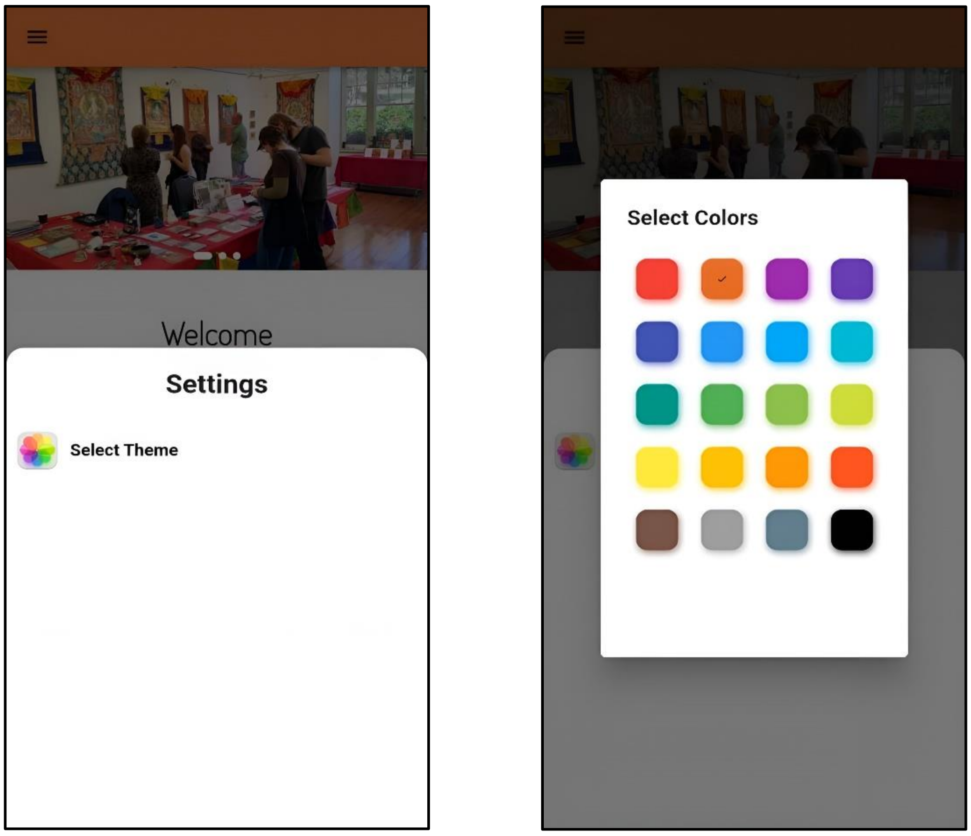
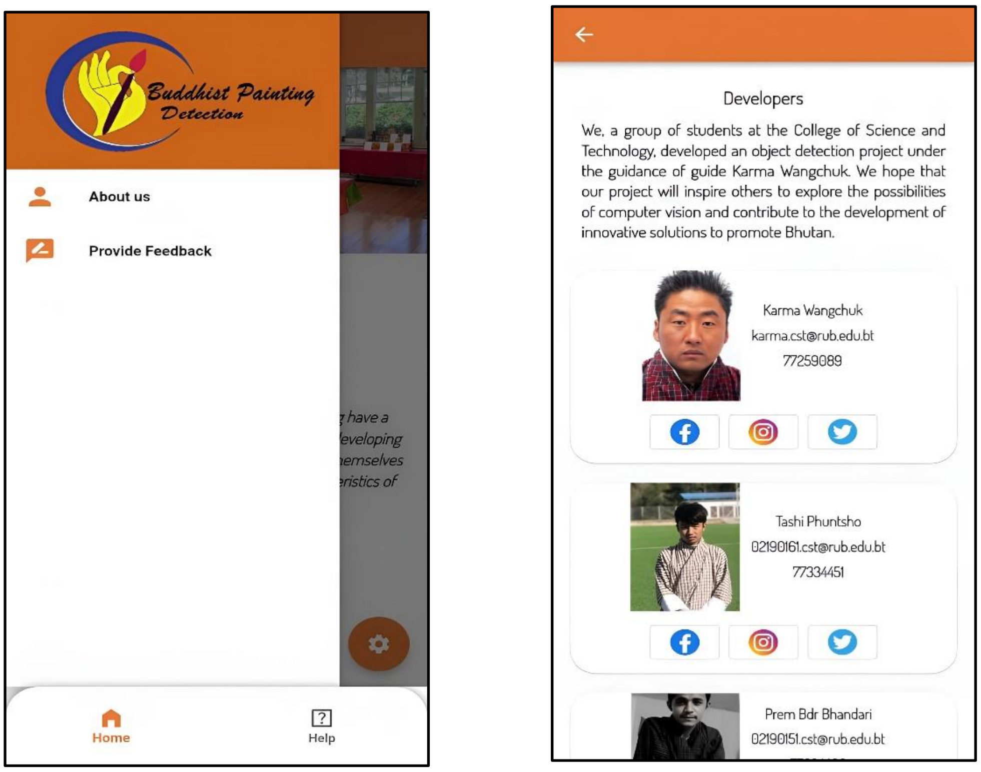


Figure 33: Mobile Settings section

Navigation Drawer and About Us page:



Developers

We, a group of students at the College of Science and Technology, developed an object detection project under the guidance of guide Karma Wangchuk. We hope that our project will inspire others to explore the possibilities of computer vision and contribute to the development of innovative solutions to promote Bhutan.

Karma Wangchuk
karma.cst@rub.edu.bt
77259089

Tashi Phuntsho
02190161.cst@rub.edu.bt
77334451

Prem Bdr Bhandari
02190151.cst@rub.edu.bt

Figure 34: Mobile Navigation drawer and about us page

Main Detection Page:

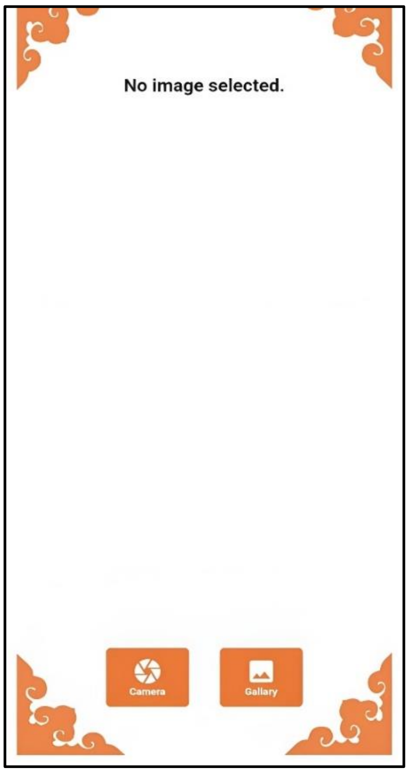


Figure 35: Mobile Detection page to input the image for detection

Detection from gallery and live detection from camera:

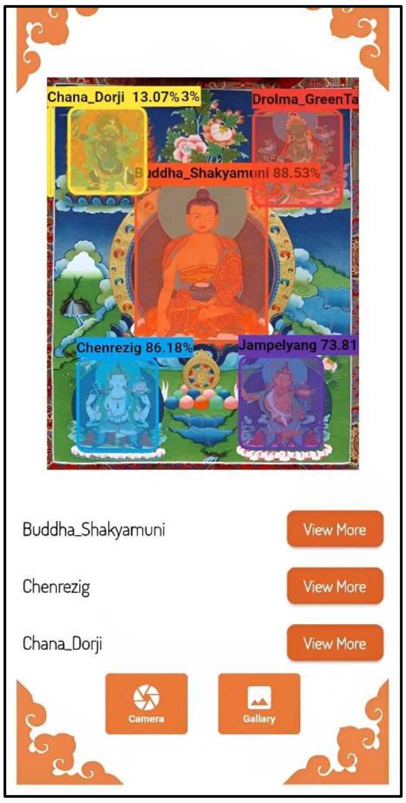
Figure 36 represents the key features and functionalities of our innovative application:

a) Object Detection in Gallery Images: Detect Buddhist paintings in images from your gallery. This functionality empowers users to analyze and identify paintings within their existing image collection. To perform object detection, users simply need to select an image from their gallery within the application.

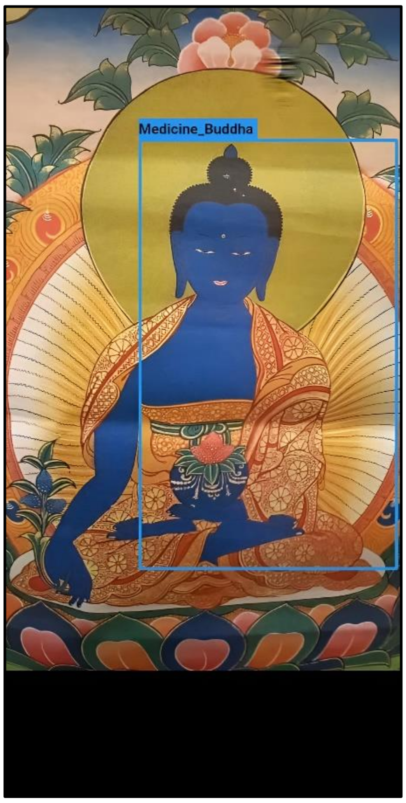
b) Live Object Detection using Camera: Application takes object detection a step further by enabling live detection using the device's camera. This exciting feature allows users to experience real-time object recognition by simply pointing their camera at a painting or any object in their surroundings.

c) Detailed Information and Insights: Upon successfully detecting a painting utilizing either method above, the application offers a seamless transition to a dedicated details page.

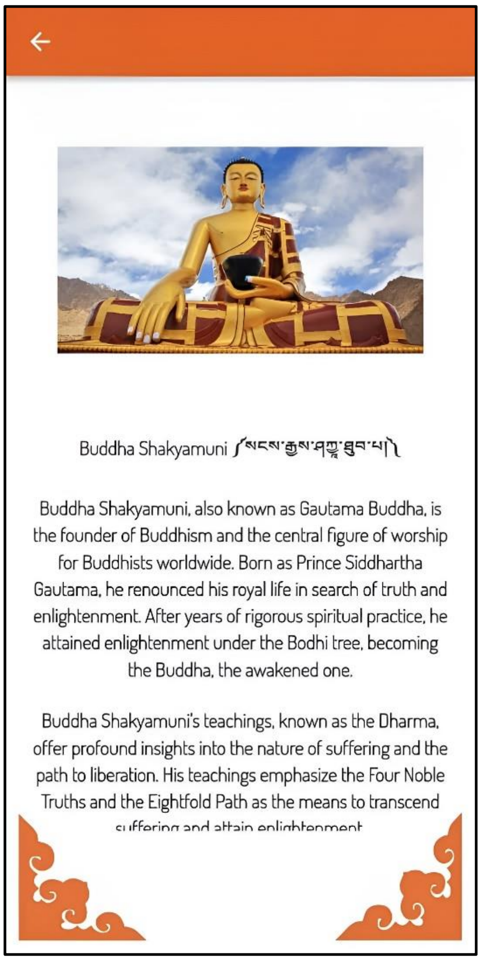
a)



b)



c)


Buddha Shakyamuni ༼སངས་རྒྱས་ཤཱཀྱ་ཐུབ་པ།༽

Buddha Shakyamuni, also known as Gautama Buddha, is the founder of Buddhism and the central figure of worship for Buddhists worldwide. Born as Prince Siddhartha Gautama, he renounced his royal life in search of truth and enlightenment. After years of rigorous spiritual practice, he attained enlightenment under the Bodhi tree, becoming the Buddha, the awakened one.

Buddha Shakyamuni's teachings, known as the Dharma, offer profound insights into the nature of suffering and the path to liberation. His teachings emphasize the Four Noble Truths and the Eightfold Path as the means to transcend suffering and attain enlightenment

Figure 36: a) Detection from gallery, b) Live detection and c) Painting description

5.8 Challenges

During our project, we encountered several challenges that impacted our progress and required careful problem solving. The major obstacle we faced was the GPU limit in Colab, Dataset Collection of Buddhist Painting Images, Low Computer Computational Power and Compatibility Issues of YOLO Model Integration

- Colab GPU Limit: Colab is a popular cloud-based development environment. Colab provides free access to GPU however, there are limitations on the duration and availability of GPU resources. This limitation hindered the training and testing processes, as the project required significant computational power. To overcome this challenge, alternative options were explored, such as using multiple google accounts.
- Dataset Collection of Buddhist Painting Images: Another challenge encountered was the collection of a suitable dataset of Buddhist painting images. Collecting a diverse and representative dataset of paintings was time consuming and challenging due to factors such as copyright restrictions, limited availability, and variations in image quality. Efforts were made to search data from various Buddhist paintings shops around Bhutan and other online repositories.
- Low Hardware Computational Power: The project faced limitations in terms of computer computational power. Training deep learning models, especially on large datasets, requires substantial computing resources, including CPU and RAM. Insufficient computational power has significantly impacted training time and model performance. To address this challenge, we explored options like google Collab, creating multiple google accounts and optimizing the model architecture and training process to reduce resource requirements.
- Compatibility Issues of YOLO Model Integration: Integrating the YOLO (You Only Look Once) model into the mobile application posed compatibility issues. YOLO is a popular object detection algorithm, but its implementation and most dependencies have been outdated and these inconsistencies have led to compatibility issues with existing code, resulting in errors. Overcoming this challenge required carefully managing dependencies, ensuring compatibility between the YOLO model and the project's existing software stack, and potentially adapting the code to address any discrepancies.

CHAPTER 6: CONCLUSION

Buddhist spiritual beings hold significant cultural and spiritual importance, yet both tourists and locals often encounter challenges in recognizing and understanding Buddhist paintings. To solve this problem, we recognized the need for a Buddhist painting recognition system. To commence our efforts, prepared a Buddhist painting dataset comprising 20 distinct classes. Leveraging the power of the VGG-16 based architecture, trained machine learning model, achieving an impressive accuracy of 91%. This initial success propelled us forward in our pursuit of an effective solution.

However, as we progress deeper into our project, we realized that a CNN-based model alone might not be optimal for handling the complexities of multiple object detection in Buddhist paintings. Undeterred by this realization, we embarked on a new approach. We assembled a specialized dataset for YOLO (You Only Look Once) and proceeded to train the YOLOv5s model. Through training and fine-tuning, we attained an accuracy of 82%. Building upon this accomplishment, we seamlessly integrated the YOLO model into our Flutter mobile application, providing users with an enhanced experience in recognizing and appreciating Buddhist paintings.

The successful integration of the YOLO model marked a significant milestone in our quest to develop a robust Buddhist painting recognition system. By harnessing the power of YOLO's advanced object detection capabilities, we were able to overcome the limitations of the CNN-based model and enhance the accuracy and efficiency of our solution. Our Flutter mobile application will empower both tourists and locals to easily identify and engage with Buddhist paintings, fostering a deeper understanding and appreciation of their spiritual significance.

REFERENCE

Alake, R. (2021, December 15). Implementing AlexNet CNN Architecture Using TensorFlow 2.0+ and Keras. *Medium*. https://towardsdatascience.com/implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A. Q., Duan, Y., Al-Shamma, O., Santamaría, J. V. G., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1). https://doi.org/10.1186/s40537-021-00444-8

Bhatt, D., Patel, C. J., Talsania, H. N., Patel, J., Vaghela, R., Pandya, S., Modi, K., & Ghayvat, H. (2021). CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope. *Electronics*, *10*(20), 2470. https://doi.org/10.3390/electronics10202470

Bochkovskiy, A., Wang, C.-Y., & Mark Liao, H.-Y. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *YOLOv4: Optimal Speed and Accuracy of Object Detection*.

Chauhan, R., Ghanshala, K. K., & Joshi, R. P. (2018). Convolutional Neural Network (CNN) for Image Detection and Recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. https://doi.org/10.1109/icsccc.2018.8703316

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *HAL (Le Centre pour la Communication Scientifique Directe)*. French National Centre for Scientific Research. https://hal.science/hal-00752497

He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition*. https://doi.org/10.1109/cvpr.2016.90

Hernández-García, A., Mehrer, J., Kriegeskorte, N., König, P., & Kietzmann, T. C. (2018). *Deep neural networks trained with heavier data augmentation learn features closer to representations in hIT*. https://doi.org/10.32470/ccn.2018.1046-0

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1412.6980

Krishna, M. M., Neelima, M., Harshali, M., & Rao, M. V. G. (2018). Image classification using Deep learning. *International Journal of Engineering and Technology(UAE)*, *7*(2.7), 614. https://doi.org/10.14419/ijet.v7i2.7.10892

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

LeCun, Y., Bengio, Y., & Hinton, G. E. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Maas, A., Hannun, A., & Ng, A. (2013). Rectifier Nonlinearities Improve Neural Network Acoustic Models. *Rectifier Nonlinearities Improve Neural Network Acoustic Models*.

Naseer, I., Akram, S., Masood, T., Jaffar, A., Khan, M. S., & Mosavi, A. (2022). Performance Analysis of State-of-the-Art CNN Architectures for LUNA16. *Sensors*, *22*(12), 4426. https://doi.org/10.3390/s22124426

O'Shea, K., & Nash, R. R. (2015). An Introduction to Convolutional Neural Networks. *An Introduction to Convolutional Neural Networks*. https://lib-arxiv-008.serverfarm.cornell.edu/pdf/1511.08458.pdf

Prashanth, D. S., Mehta, R. V. K., & Sharma, N. (2020). Classification of Handwritten Devanagari Number – An analysis of Pattern Recognition Tool using Neural Network and CNN. *Procedia Computer Science, 167*, 2445–2457. https://doi.org/10.1016/j.procs.2020.03.297

Redmon, J., Divvala, S. K., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. https://doi.org/10.1109/cvpr.2016.91

Redmon, J., & Farhadi, A. (2017). *YOLO9000: Better, Faster, Stronger*. https://doi.org/10.1109/cvpr.2017.690

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1804.02767

Saxena, S. (2021). The Architecture of Lenet-5. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/#:~:text=It%20has%203%20convolution%20layers,of%20trainable%20parameters%20is%2060000.

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Computer Vision and Pattern Recognition*. http://export.arxiv.org/pdf/1409.1556

Tan, Y. (2022). Feature Recognition and Style Transfer of Painting Image Using Lightweight Deep Learning. *Computational Intelligence and Neuroscience*, *2022*, 1–10. https://doi.org/10.1155/2022/1478371

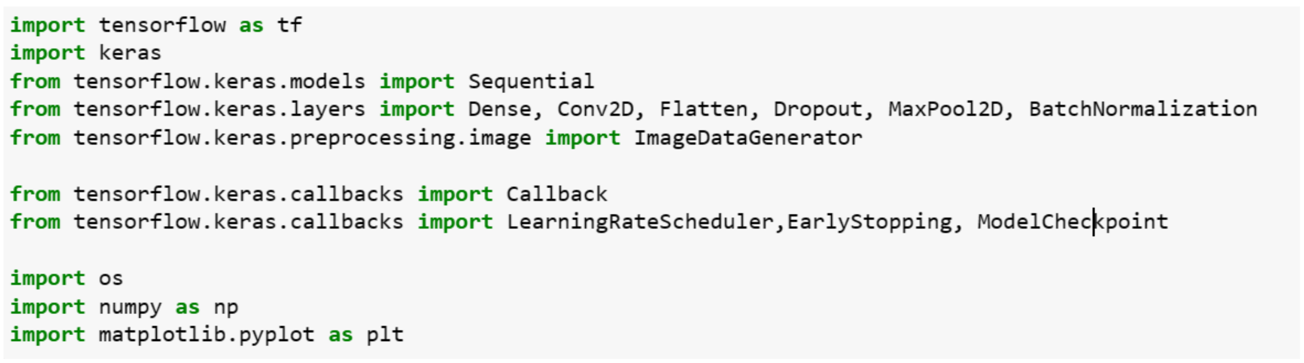
Thakur, R. (2023, May 13). Step by step VGG16 implementation in Keras for beginners. *Medium*. https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

Wang, Z., Lian, J., Song, C., Zhang, Z., Zheng, W., Yue, S., & Ji, S. (2019). SAS: Painting Detection and Recognition via Smart Art System With Mobile Devices. *IEEE Access*. https://doi.org/10.1109/access.2019.2941239

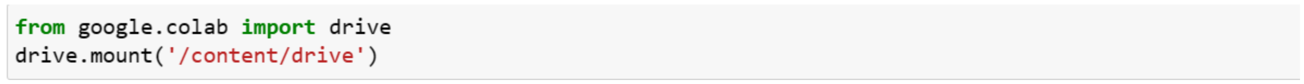
What is Deep Learning? | IBM. (n.d.). https://www.ibm.com/topics/deep-learning#:~:text=the%20next%20step-,What%20is%20deep%20learning%3F,from%20large%20amounts%20of%20data.

APPENDICES

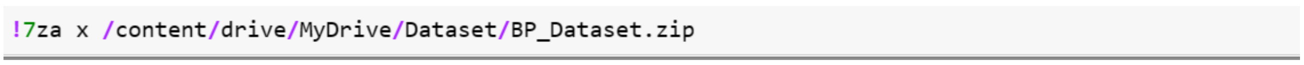
Source code of VGG-16 based architecture that was used:



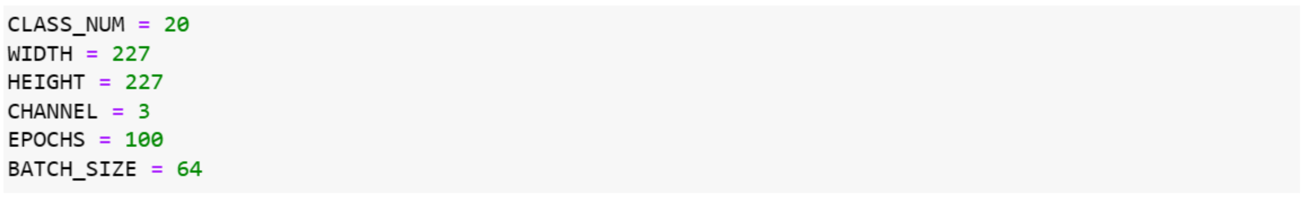
```
import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPool2D, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import LearningRateScheduler,EarlyStopping, ModelCheckpoint

import os
import numpy as np
import matplotlib.pyplot as plt
```



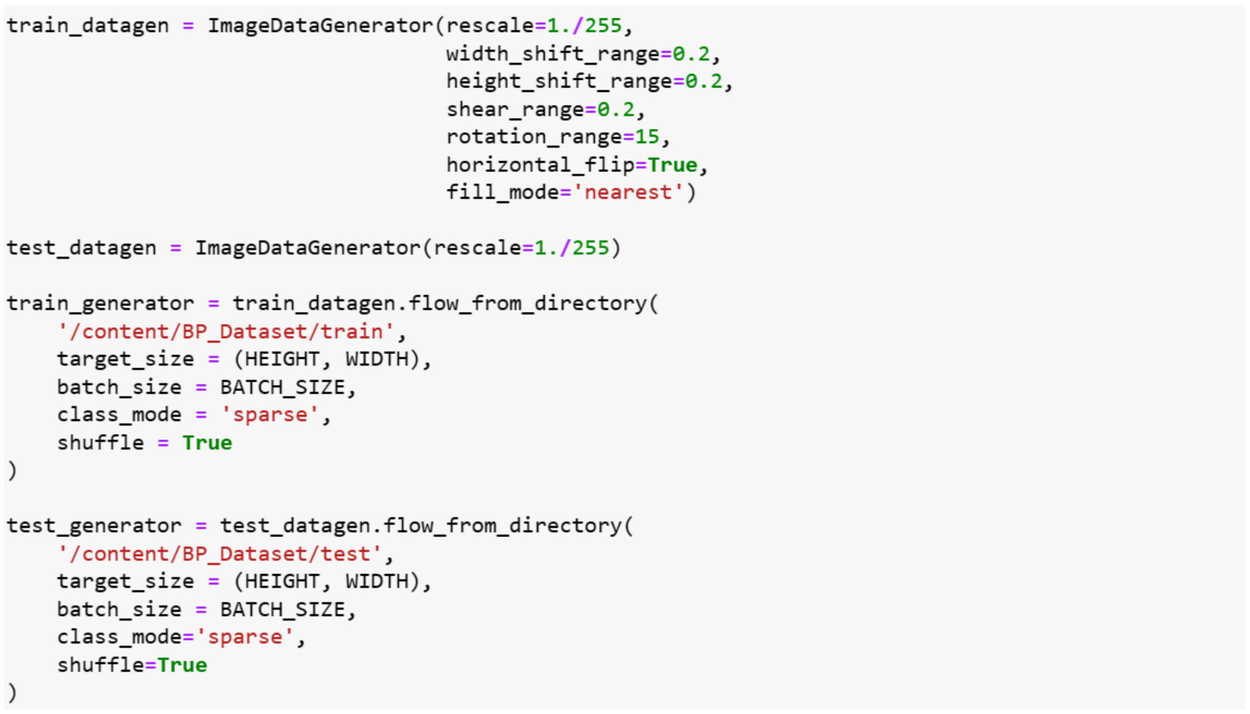
```
from google.colab import drive
drive.mount('/content/drive')
```



```
!7za x /content/drive/MyDrive/Dataset/BP_Dataset.zip
```



```
CLASS_NUM = 20
WIDTH = 227
HEIGHT = 227
CHANNEL = 3
EPOCHS = 100
BATCH_SIZE = 64
```



```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   rotation_range=15,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    '/content/BP_Dataset/train',
    target_size = (HEIGHT, WIDTH),
    batch_size = BATCH_SIZE,
    class_mode = 'sparse',
    shuffle = True
)

test_generator = test_datagen.flow_from_directory(
    '/content/BP_Dataset/test',
    target_size = (HEIGHT, WIDTH),
    batch_size = BATCH_SIZE,
    class_mode='sparse',
    shuffle=True
)
```

```
model = Sequential()

model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())
```

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(Dense(units=4096,activation="relu"))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Dense(units=5, activation="softmax"))
```

```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.h5"

# Call back
checkpoint = ModelCheckpoint(filepath, monitor = ['val_accuracy'], verbose = 1  , mode = 'max')
sheduler_lr = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
early_stop = EarlyStopping(monitor = 'val_loss', patience=5, verbose=1)

callbacks_list = [checkpoint, early_stop,sheduler_lr]

# Steps
SPE = len(train_generator)
VS = len(test_generator)
```

```
history = model.fit(
    train_generator,
    steps_per_epoch = SPE,
    validation_data = test_generator,
    validation_steps = VS,
    epochs = EPOCHS,
    callbacks = callbacks_list
)
```