SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA, INDORE
SHRI VAISHNAV INSTITUTE OF INFORMATION TECHNOLOGY DEPARTMENT OFCOMPUTER SCIENCE & ENGINEERING

JANUARY-JUNE 2021

Project Report

on

IOS OBJECT CLASSIFICATION USING ML (MACHINELEARNING) API.

In partial fulfillment of requirements for the degree

of

BACHELOR OF TECHNOLOGY
IN

COMPUTER SCIENCE & ENGINEERING

Submitted by:

DRASHTI BHASIN [19100BTCSEMA05482]

KHUSHI PRADHAN [19100BTCSEMA05486]

RAVNISH SINGH [19100BTCSEMA05495]

Under the guidance of

MR. NITIN RATHORE

SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA, INDORE
SHRI VAISHNAV INSTITUTE OF INFORMATION TECHNOLOGY DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

JANUARY-JUNE 2022

SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA, INDORE
SHRI VAISHNAV INSTITUTE OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DECLARATION

We here declare that work which is being presented in the project entitled "**IOS FACE DETECTION AND OBJECT CLASSIFICATION USING ML (MACHINE LEARNING) API"** in partial fulfillment of degree of **Bachelor of Technology in Computer Science & Engineering** is an authentic record of our work carried out under the supervision and guidance of **Mr. NITIN RATHORE** Asst. Professor of Computer Science & Engineering. The matter embodied in this project has not been submitted for the award of any other degree.

DRASHTI BHASIN

KHUSHI PRADHAN

RAVNISH SINGH

Date:

SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA, INDORE

SHRI VAISHNAV INSTITUTE OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PROJECT APPROVAL SHEEET

Following team has done the appropriate work related to the "**IOS OBJECT CLASSIFICATION USING ML (MACHINE LEARNING) API**" in partial fulfillment for the award of **Bachelor of Technology in Computer Science & Engineering** of "SHRI VAISHNAV INSTITUTE OF INFORMATION TECHNOLOGY" and is being submittedto SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA, INDORE.

Team:
1. DRASHTI BHASIN [19100BTCSEMA05482]
2. KHUSHI PRADHAN [19100BTCSEMA05486]
3. RAVNISH SINGH [19100BTCSEMA05495]

Internal Examiner **External Examiner**

Date

SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA, INDORE
SHRI VAISHNAV INSTITUTE OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that **Ms. DRASHTI BHASIN, Ms. KHUSHI PRADHAN and Mr. RAVNISH SINGH** working in a team have satisfactorily completed the project entitled "**PROJECT TITLE**" under the guidance of **Mr. NITIN RATHORE** in the partial fulfillment of the degree of **Bachelor of Technology in Computer Science & Engineering** awarded by SHRI VAISHNAV INSTITUTE OF INFORMATION TECHNOLOGY affiliated to SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA, INDORE during the academic year **January 2022-June 2022**.

MR. NITIN RATHORE
Project Guide

Dr. Anand Rajavat
Director & Head,
Department of Computer Science & Engineering

ACKNOWLEDGEMENT

We are grateful to several persons for their advice and support during the time of complete our project work. First and foremost, our thanks goes to **Dr. Anand Rajavat** Head of the Department of Computer Science & Engineering and **Mr. NITIN RATHORE** the mentor of our project for providing us valuable support and necessary help whenever required and also helping us explore new technologies by the help of their technical expertise. His direction, supervision and constructive criticism were indeed the source of inspiration for us.

We would also like to express our sincere gratitude towards our Director **Dr. Anand Rajavat** for providing us valuable support.

We also owe our sincere thanks to all the **faculty members** of Computer Science & Engineering Department who have always been helpful.

We forward our sincere thanks to all **teaching and non-teaching staff** of Computer Science & Engineering department, SVVV Indore for providing necessary information and their kind co-operation.

We would like to thank our parents and family members, our classmates and our friends for their motivation and their valuable suggestion during the project. Last, but not the least, we thank all those people, who have helped us directly or indirectly in accomplishing this work. It has been a privilege to study at SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA, INDORE

ABSTRACT

The main aim of this project is to build a system that detects objects from the image or a stream of images given to the system in the form of previously recorded video or the real time input from the camera. Bounding boxes will be drawn around the objects that are being detected by the system. The system will also classify the object to the classes the object belongs. Swift Programming and a Machine Learning Technique named using Core ML (Machine Learning).

Image detection. That's where an app can detect an image, and then respond when the iOS device camera recognizes that image in the real world. Image detection works with two-dimensional items such as pictures and photographs.

The app in this sample identifies the most prominent object in an image by using MobileNet, an open-source image classifier model that recognizes around 1,000 different categories.

Each time a user selects a photo from the library or takes a photo with a camera, the app passes it to a Vision image classification request. Vision resizes and crops the photo to meet the MobileNet model's constraints for its image input, and then passes the photo to the model using the Core ML framework behind the scenes. Once the model generates a prediction, Vision relays it back to the app, which presents the results to the user.

The sample uses MobileNet as an example of how to use a third-party Core ML model. You can download open-source models — including a newer version of MobileNet — on the Core ML model gallery.

Before you integrate a third-party model to solve a problem — which may increase the size of your app — consider using an API in the SDK. For example,
the Vision framework's VNClassifyImageRequest class offers the same functionality as MobileNet, but with potentially better performance and without increasing the size of your app (see Classifying Images for Categorization and Search).

To take photos within the app, run the sample on a device with a camera. Otherwise, you can select photos from the library in Simulator.

List of Figures

TABLE OF CONTENT

CHAPTER-1

INTRODUCTION

1.1 Introduction-

The app in this sample identifies the most prominent object in an image by using MobileNet, an open source image classifier model that recognizes around 1,000 different categories.

FIGURE -1.1

Each time a user selects a photo from the library or takes a photo with a camera, the app passes it to a Vision image classification request.

Vision resizes and crops the photo to meet the MobileNet model's constraints for its image input, and then passes the photo to the model using the Core ML framework behind the scenes. Once the model generates a prediction, Vision relays it back to the app, which presents the results to the user.

The sample uses MobileNet as an example of how to use a third-party Core ML model. You can download open source models — including a newer version of MobileNet — on the Core ML model gallery.

Before you integrate a third-party model to solve a problem — which may increase the size of your app — consider using an API in the SDK.

For example, the Vision framework's VNClassifyImageRequest class offers the same functionality as MobileNet, but with potentially better performance and without increasing the size of your app (see Classifying Images for Categorization and Search).

Also you can make a custom image classifier that identifies your choice of object types with Create ML. See Creating an Image Classifier Model to learn how to create a custom image classifier that can replace the MobileNet model in this sample.

1.2 Problem Statement -

The project "Object Detection System using Machine Learning Technique" detects objects efficiently based on the algorithm on image data and video data to detect objects.

1. Dual priorities: object classification and localization
The first major complication of object detection is its added goal: not only do we want to classify image objects but also to determine the objects' positions, generally referred to as the *object localization* task. To address this issue, researchers most often use a multi-task loss function to penalize both misclassifications and localization errors.

2. Speed for real-time detection
Object detection algorithms need to not only accurately classify and localize important objects, they also need to be incredibly fast at prediction time to meet the real-time demands of video processing. Several key enhancements over the years have boosted the speed of these algorithms, improving test time from the 0.02 frames per second (fps) of R-CNN to the impressive 155 fps of Fast YOLO.

3. Multiple spatial scales and aspect ratios
For many applications of object detection, items of interest may appear in a wide range of sizes and aspect ratios. Practitioners leverage several techniques to ensure detection algorithms are able to capture objects at multiple scales and views.

4. Limited data
The limited amount of annotated data currently available for object detection proves to be another substantial hurdle. Object detection datasets typically contain ground truth examples for about a dozen to a hundred classes of objects, while image classification datasets can include upwards of 100,000 classes. Furthermore, crowdsourcing often produces image classification tags for free (for example, by parsing the text of user-provided photo captions). Gathering ground truth labels *along with* accurate bounding boxes for object detection, however, remains incredibly tedious work.

5. Class imbalance
Class imbalance proves to be an issue for most classification problems, and object detection is no exception. Consider a typical photograph. More likely than not, the photograph contains a few main objects and the remainder of the image is filled with background. Recall that selective search in R-CNN produces 2,000 candidate RoIs per image–just imagine how many of these regions do not contain objects and are considered negatives!

1.3 Need for the proper System -

Following are the requirement-

1. IOS 11 or later
2. MacOS 10.13 or later
3. Xcode latest version
4. To take photos within the app, run the sample on a device with a camera. Otherwise, you can select photos from the library in Simulator.

1.4 Objective -

The Objective of the project is to Crop and scale photos using the Vision framework and classify them with a Core ML model and to take photos within the app, run the sample on a device with a camera. Otherwise, selected photos from the library in Simulator.

1.5 Modules of the system -

Modularisation of this IOS app is done using-

1. Dynamic frameworks

A framework is a structured in directory that can contain shared code and shared resources such images, nibs (compiled form of xibs and storyboards) and other assets.

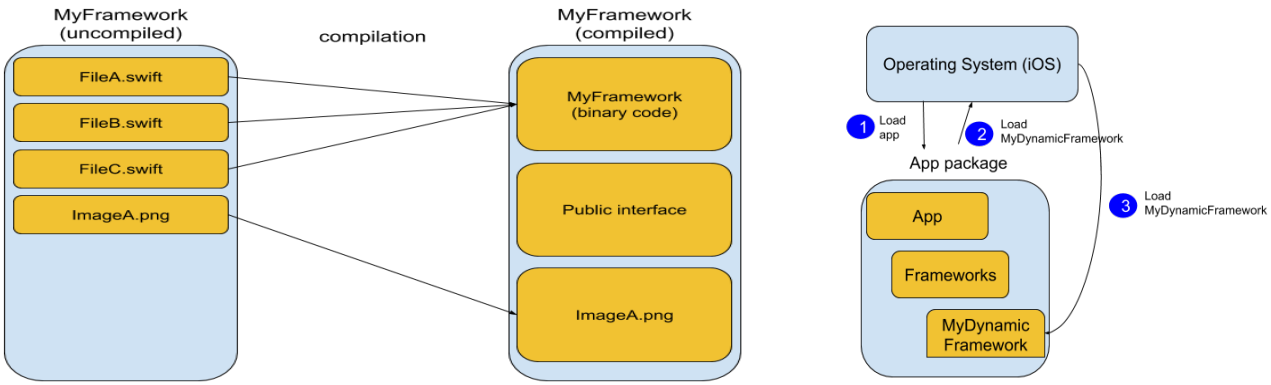


FIGURE -1.2

2. Static libraries

A static library is a collection of compiled source code files. Let's say we have FileA.swift, FileB.swift and FileC.swift. With a static library we can compile these files and wrap them inside a single file containing all of these. The file has has an extension of .a; short for archive. Sort of like getting some pages and making a book out of it.

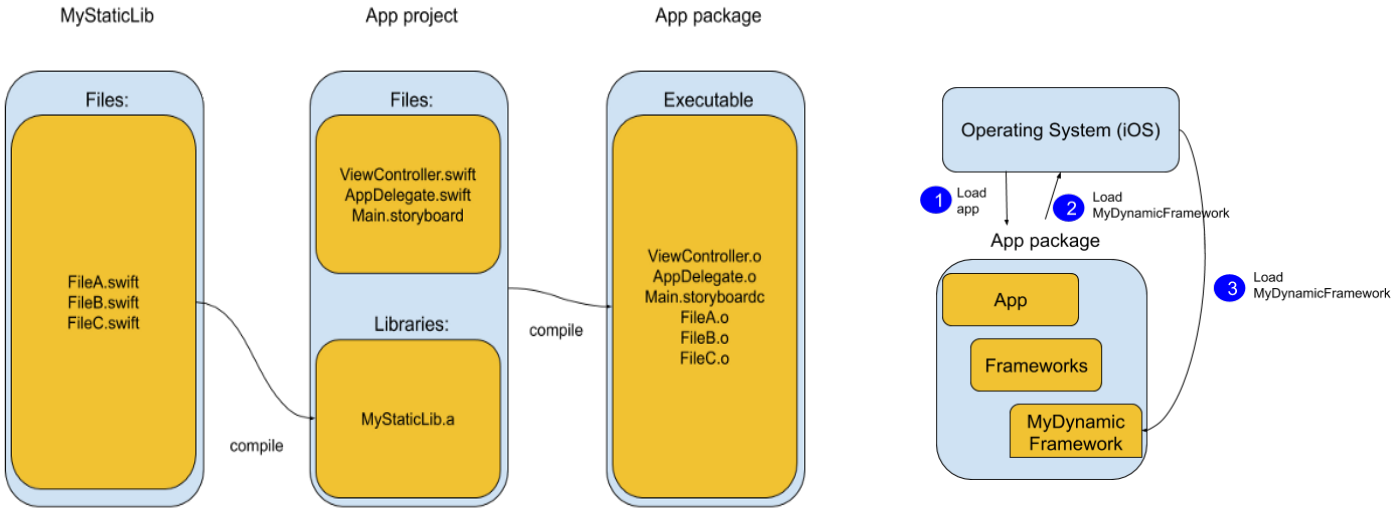


FIGURE -1.3

1.6 Scope –

Object detection is a vision technique that **allows us to identify and locate objects in an image or video**. With this kind of identification and localization, object detection can be used to count objects in a scene and determine and track their precise locations, all while accurately labelling them.

CHAPTER-2
LITERATURE SURVEY

2.1 Existing System -

The situation has been significantly changed with the advent of the operating system when the majority of the developers switched to solving the problems of image processing itself. However, this has not yet led to the cardinal progress in solving typical tasks of recognizing faces, car numbers, road signs, analyzing remote and medical images, etc. Each of these "eternal" problems are solved by trial and error by the efforts of numerous groups of the engineers and scientists.

As modern technical solutions are turn out to be excessively expensive, the task of automating the creation of the software tools for solving intellectual problems is formulated and intensively solved abroad. In the field of image processing, the required tool kit should be supporting the analysis and recognition of images of previously unknown content and ensure the effective development of applications by ordinary programmers. Just as the Windows toolkit supports the creation of interfaces for solving various applied problems. Object recognition is to describe a collection of related computer vision tasks that involve activities like identifying objects in digital photographs. Image classification involves activities such as predicting the class of one object in an image.
Object localization is referring to identifying the location of one or more objects in an image and drawing an abounding box around their extent. Object detection does the work of combines these two tasks and localizes and classifies one or more objects in an image. When a user or practitioner refers to the term "object recognition", they often mean "object detection".

2.2 Proposed System -

With reduced complexity, faster computation, latest dynamic framework, and static libraries this system provides a means to crop and scale photos using the Vision framework and classify them with a Core ML model and to take photos within the app, run the sample on a device with a camera. Otherwise, selected photos from the library in Simulator.

2.3 Feasibility Study -

Overall App Feasibility-
• Considering the major features of the current scope (at a high level) and evaluating as they are feasible
• Benefits of the product or service
– Will earn money from downloads (APP STORE).
– Will move one step ahead in development like camera and use of AR, VR.
• Old concept: App in which most of the technical concepts has already been implemented earlier in our apps or other apps in market like Photo editor, Identify screen etc.

2.3.1 Technical Feasibility-

- Experimental features: Identified features in the design that seem experimental in nature, such as techniques, perspectives, or other unique ideas.
- Classifying *and* finding an unknown number of individual objects within an image, however, was considered an extremely difficult problem only a few years ago.
- The **feature pyramid network (FPN)** takes the concept of multiple feature maps one step further. Images first pass through the typical CNN pathway, yielding semantically rich final layers.

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embedding and segmentation similar to how other popular large scale models. MobileNet has **accuracy 65% in 100 epochs**. But as we can see in the training performance of MobileNet, its accuracy is getting improved and it can be inferred that the accuracy will certainly be improved if we run the training for more number of epochs.

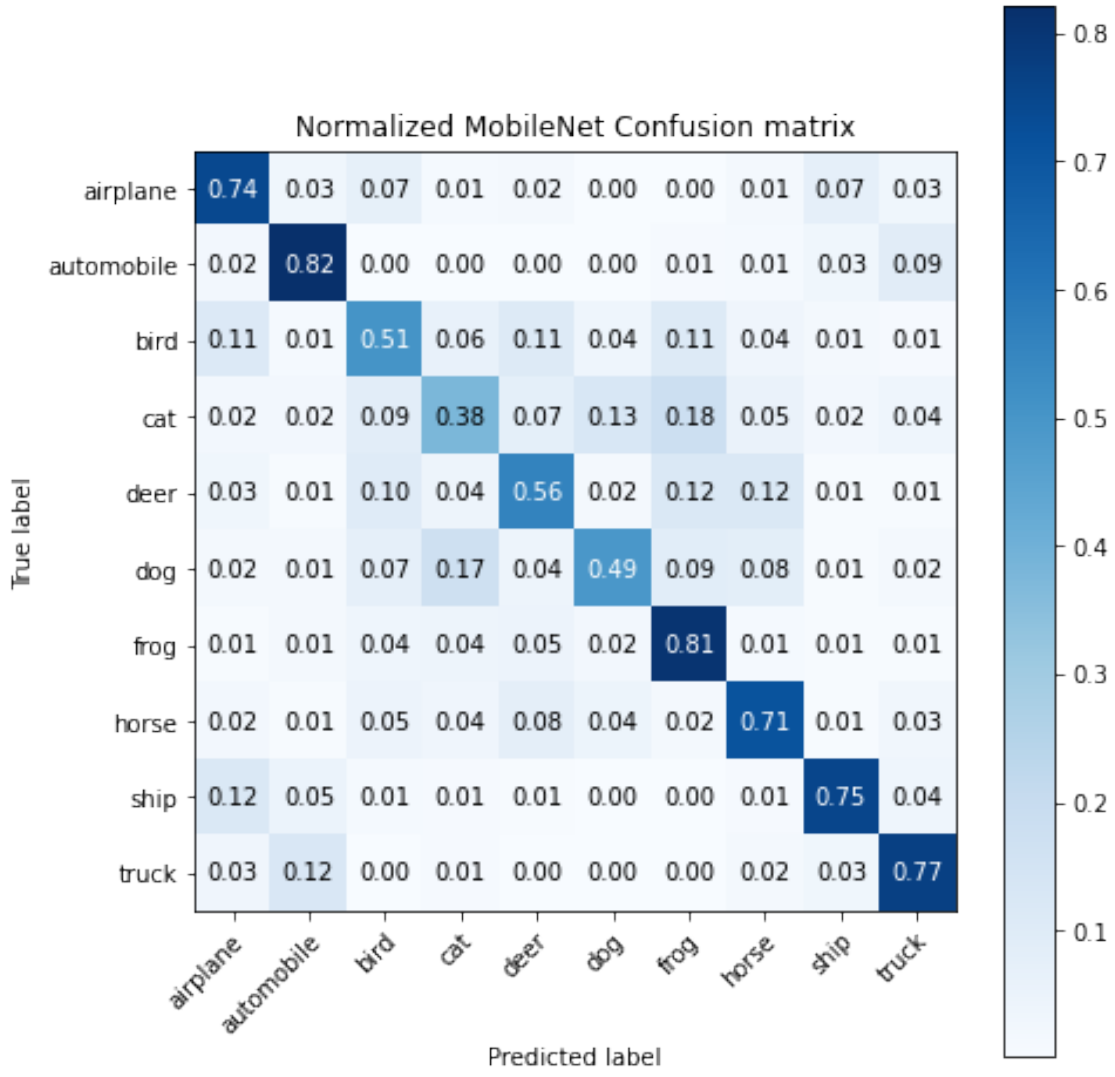


FIGURE -2.1

Machine Learning Image Analysis
Training a Create ML Model to Classify Flowers
Train a flower classifier using Create ML in Swift Playgrounds, and apply the resulting model to real-time image classification using Vision.
class VNCoreMLRequest
An image analysis request that uses a Core ML model to process images.
class VNClassificationObservation
An object that represents classification information that an image analysis request produces.
class VNPixelBufferObservation
An object that represents an image that an image analysis request produces.
class VNCoreMLFeatureValueObservation
An object that represents a collection of key-value information that a Core ML image analysis request produces.

• What kinds of technology will we need?

1. IOS 11 or later
2. MacOS 10.13 or later
3. Xcode latest version

4. To take photos within the app, run the sample on a device with a camera. Otherwise, you can select photos from the library in Simulator.

• Is relevant technology developed enough to be easily applied to our problem?
Yes, who are you having ready when technology developed enough to be applied and get thorough solution but in the future scope we need to incorporate technologies which are developing to their fullest potential such as augmented reality and virtual reality.
• Do we currently possess the necessary technology?
- Yes, and it does have the capacity to handle the solutions around 1,000 different categories of open-source image classifier model.

MobileNet uses **depthwise separable convolutions.** It significantly **reduces the number of parameters** when compared to the network with regular convolutions with the same depth in the nets. This results in lightweight deep neural networks.
A depthwise separable convolution is made from two operations.

1. **Depthwise convolution.**
2. **Pointwise convolution**.

MobileNet is a class of CNN that was open-sourced by Google, and therefore, this gives us an excellent starting point for training our classifiers that are insanely small and insanely fast.

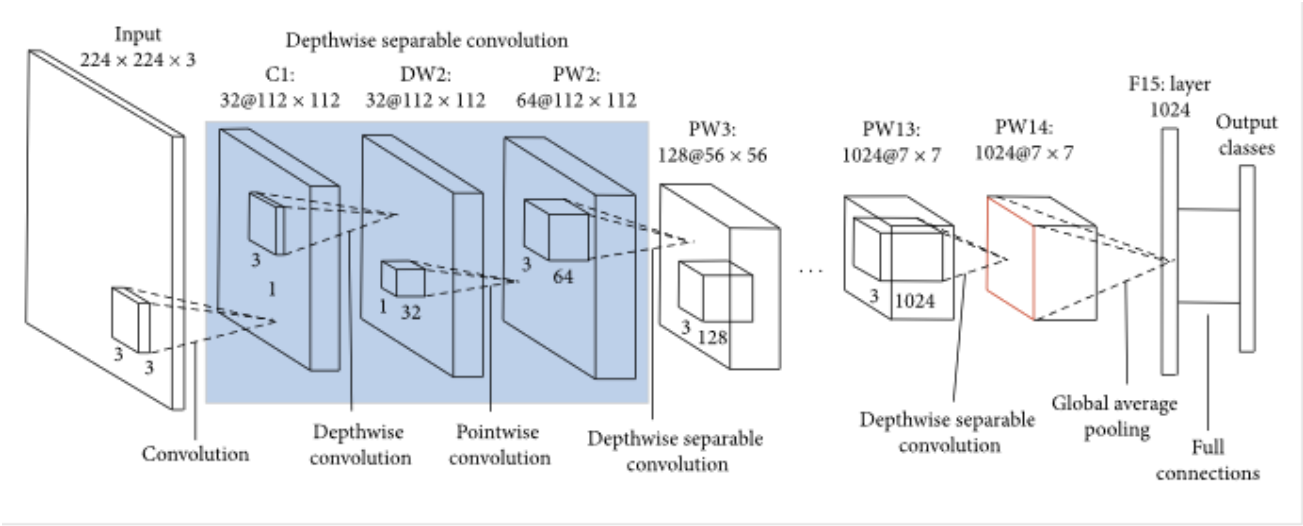


FIGURE -2.2

MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

FIGURE -2.3

2.3.2 Economic Feasibility-

• Resource cost is based on the estimated resources within the technical analysis.
• Employee costs should be based on salaries and overhead.
  - In the event of expansion as automated technology require this application to help automate tasks.
• Any hardware or software that you purchase should be listed as well.
  - Currently none as it uses free API and frameworks provided by Core ML(Machine Learning) from Apple Development.
• Additional costs (if any): This section is an assessment of additional costs incurred from licensing, contracting, out-sources testing, and so on.
  - None.
• Cost of maintenance of equipment.

- MobileNet Structure

- The standard convolution has the computation cost of

- $D_k \cdot D_k \cdot M \cdot N \cdot D\_F \cdot D\_F$

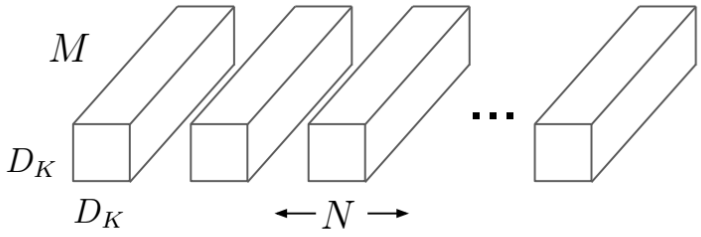
- Depthwise separable convolution costs

- $D_k \cdot D_k \cdot M \cdot D\_F \cdot D\_F + M \cdot N \cdot D\_F \cdot D\_F$

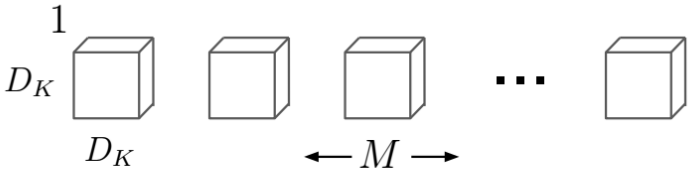
2.3.3 Operational Feasibility-

• Assessment of the overall appeal to the market for the APP
• Market timeliness: best suitable time for release
• Identification of the target audience
  - Cloud based platforms, Automation Technology etc.
• Comparing with other similar competing Apps.

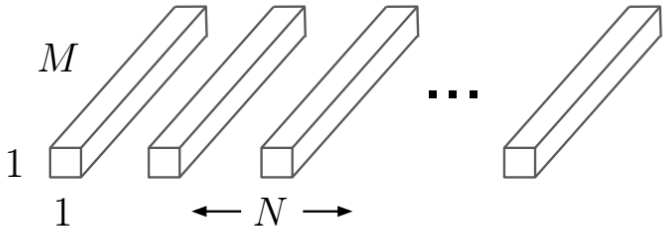
Depthwise separable convolutions which is a form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a 1×11×1 convolution called a pointwise convolution. In MobileNet, the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×11×1 convolution to combine the outputs the depthwise convolution. The following figure illustrates the difference between standard convolution and depthwise separable convolution.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

FIGURE -2.4

CHAPTER-3
REQUIREMENTS ANALYSIS

3.1 Method used for Requirement analysis -

Data flow diagram
This technique is used to visually represent systems and processes that are complex and difficult to describe in text. Data flow diagrams represent the flow of information through a process or a system. It also includes the data inputs and outputs, data stores, and the various subprocess through which the data moves. DFD describes various entities and their relationships with the help of standardized notations and symbols. By visualizing all the elements of the system, it is easier to identify any shortcomings. These shortcomings are then eliminated in a bid to create a robust solution.

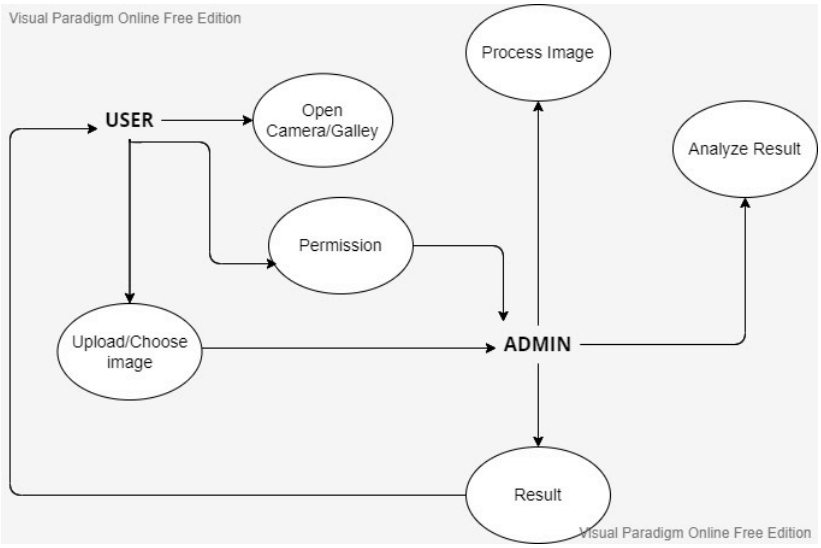


FIGURE -3.1

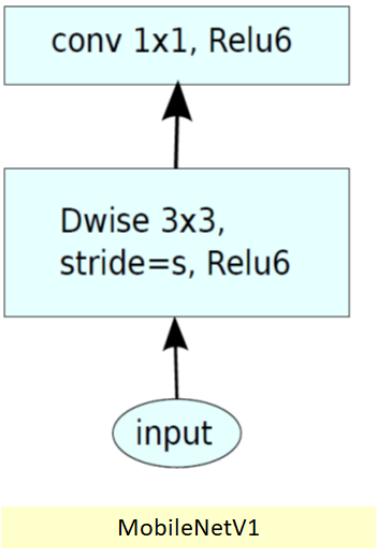


FIGURE -3.2

3.2 Functional Requirements –

Functions the product is required to perform-
The project needs to Crop and scale photos using the Vision framework and classify them with a Core ML model and to take photos within the app, run the sample on a device with a camera. Otherwise, selected photos from the library in Simulator.

The method creates a Core ML model instance for Vision by:

- Creating an instance of the model's wrapper class that Xcode auto-generates at compile time.
- Retrieving the wrapper class instance's underlying MLModel property.
- Passing the model instance to a VNCoreMLModel initializer.
- The Image Predictor class minimizes runtime by only creating a single instance it shares across the app.

Image Recognition plays an important role in many fields like medical disease analysis, and many more furthermore we will mainly focus on how to Recognize the given image, what is being displayed.

3.4 Non-Functional Requirements -

1: Identify Key Stakeholders and End-Users

The first step of the requirements analysis process is to identify key stakeholders who are the main sponsors of the project. They will have the final say on what should be included in the scope of the project.

Next, identify the end-users of the product. Since the product is intended to satisfy their needs, their inputs are equally important.

2: Capture Requirements

Ask each of the stakeholders and end-users their requirements for the new product. Here are some requirements analysis techniques that you can use to capture requirements:

- Hold One-on-One Interviews

Interview each stakeholder and end-user individually. This technique will help you gather specific requirements.

- Use Focus Groups

Conduct group interviews or group workshops to understand the flow of information between different stakeholders and end-users. This technique will ensure that there will be no conflict of interest later on during the project.

-Utilize Use Cases

Use cases provide a walkthrough of the entire product through the eyes of the end-user. This technique will help visualize how the product will actually work.

- Build Prototypes

A prototype provides users a sample look and feel of the final product. This technique will help address feasibility issues and identify problems ahead of time.

3.3 System Specification –

3.5.1 Hardware specification-

This app targets iOS operating system. A list of all the current devices supporting this OS:
iPhone:11 or later
MacOS 10.13 or later
iPod: 5th generation.
Pad: 2, 3rd and 4th generation, Air.
The app has been tested on the above highlighted devices. However, all the performance metrics and exhaustive tests have been executed on the iPhone 13. This device presents the following technical specifications:
Processor 1.3 GHz dual-core Apple-designed ARMv7s (Apple A6). RAM 1GB LPDDR2 DRAM.
Graphics PowerVR SGX543MP3 (tri-core, 266 MHz) GPU. Storage 16, 32 or 64 GB.
Back Camera 8 MP photos, f/2.4, 1080p HD video (30 fps), Infrared cut- o↵ filter, Back-illuminated sensor, face detection, video stabilization, panorama and ability to take photos while shooting videos.
Front Camera 1.2 MP photos, 720p HD video (30 fps), Back-illuminated sensor.
For the preset we use, the camera uses images with a resolution of 460 ⇥ 380.

3.5.2 Software Specification-

- Xcode: Latest version.
- Vision Framework.
- IOS: 11 or later.
- Core ML Model.

CHAPTER-4
DESIGN

4.1 Software Requirements Specification –

- Xcode: Latest version.
- Vision Framework.
- IOS: 11 or later.
- Core ML Model.

4.1.1 Supplementary Specifications -

iPhone:11 or later
MacOS 10.13 or later
iPod: 5th generation.
Pad: 2, 3rd and 4th generation, Air.
The app has been tested on the above highlighted devices. However, all the performance metrics and exhaustive tests have been executed on the iPhone 13. This device presents the following technical specifications:
Processor 1.3 GHz dual-core Apple-designed ARMv7s (Apple A6). RAM 1GB LPDDR2 DRAM.
Graphics PowerVR SGX543MP3 (tri-core, 266 MHz) GPU. Storage 16, 32 or 64 GB.
Back Camera 8 MP photos, f/2.4, 1080p HD video (30 fps), Infrared cut- o↵ filter, Back-illuminated sensor, face detection, video stabilization, panorama, and ability to take photos while shooting videos.
Front Camera 1.2 MP photos, 720p HD video (30 fps), Back-illuminated sensor.
For the preset we use, the camera uses images with a resolution of 460 ⇥ 380.

4.1.1 Glossary -

ML	Machine Learning
CNN	MobileNet **Convolutional neural network** Machine Learning Algorithms
MAC's	*Multiply-Accumulates*
TC's	Test Cases
DFD	Data Flow Diagram

4.1.2 Use Case Model –

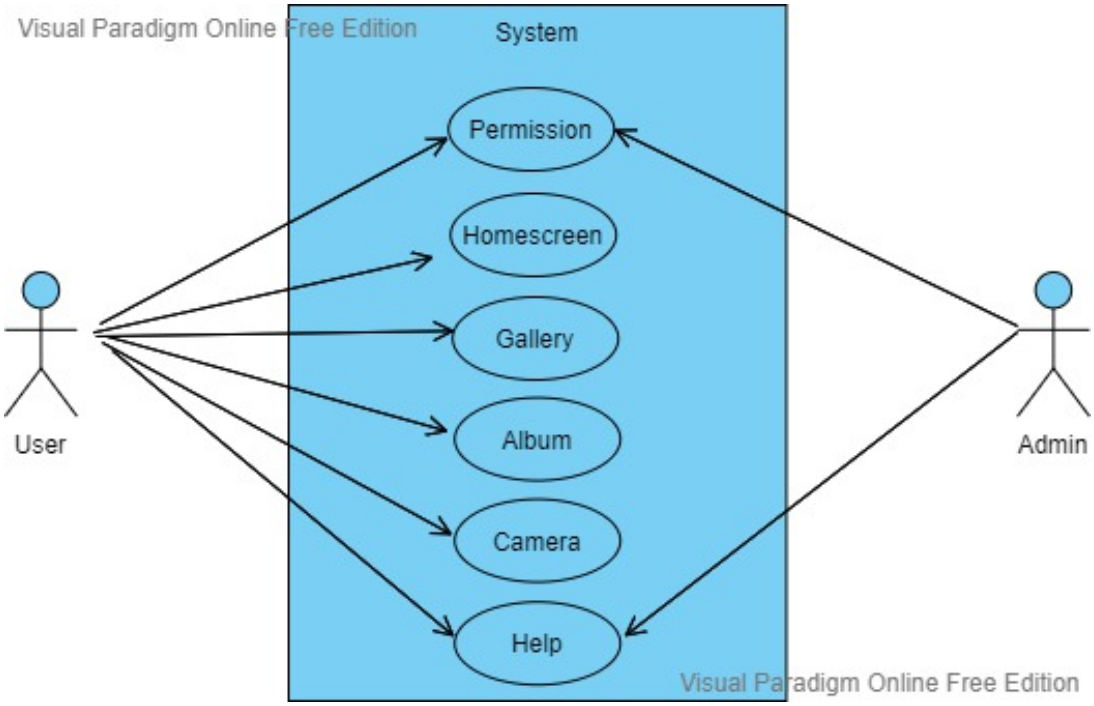


FIGURE -4.1

4.2 Data flow Diagram (Level 0,1)

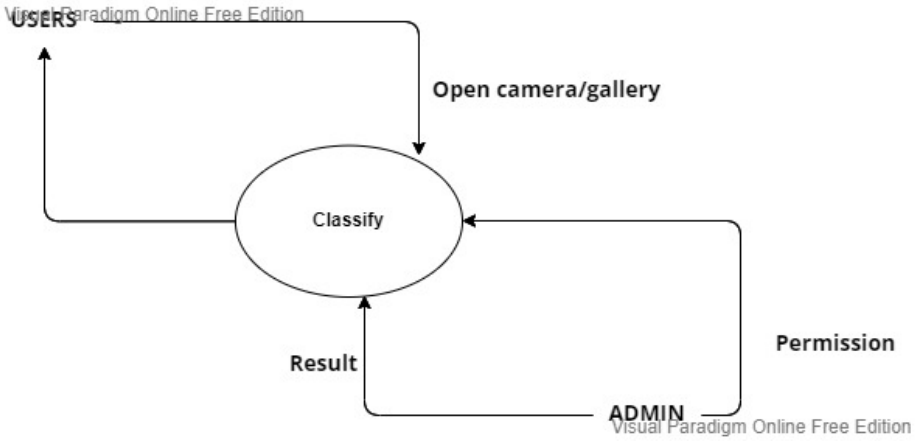


FIGURE -4.2

DFD Level-0

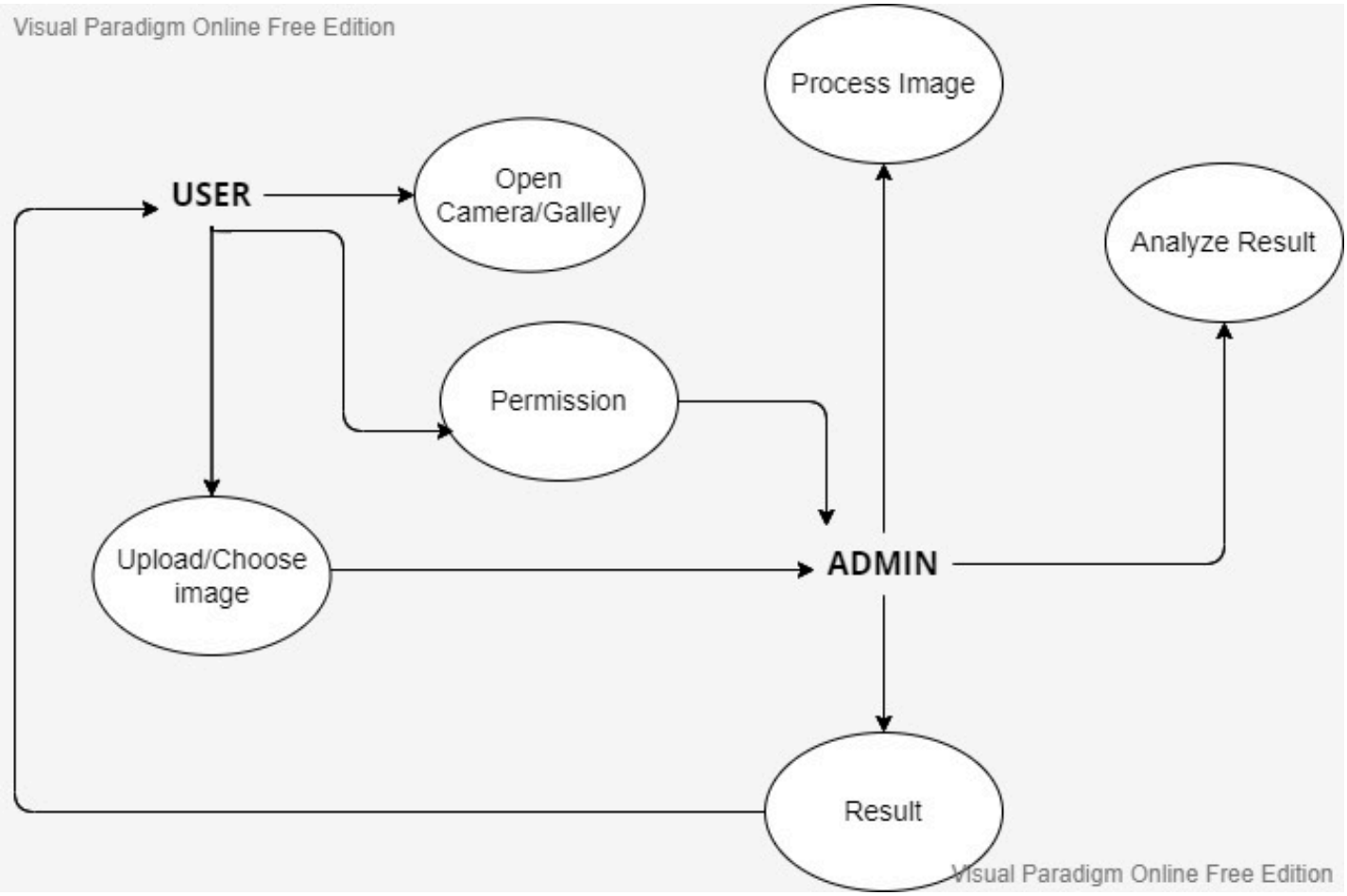


FIGURE -4.3

DFD Level-1

4.3 Database Design (ER-Diagram)-

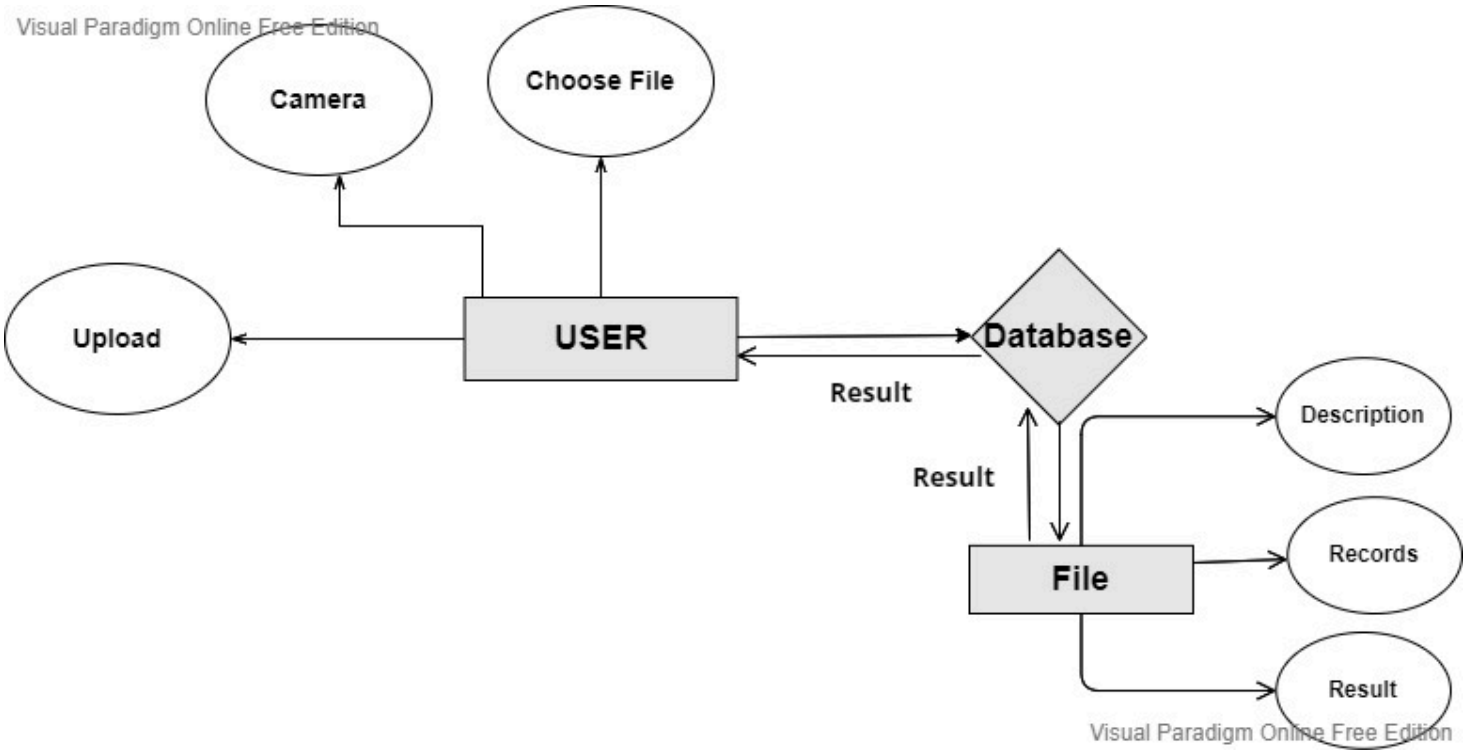


FIGURE -4.4

CHAPTER-5
SYSTEM MODELING

5.1 Detailed Class Diagram-

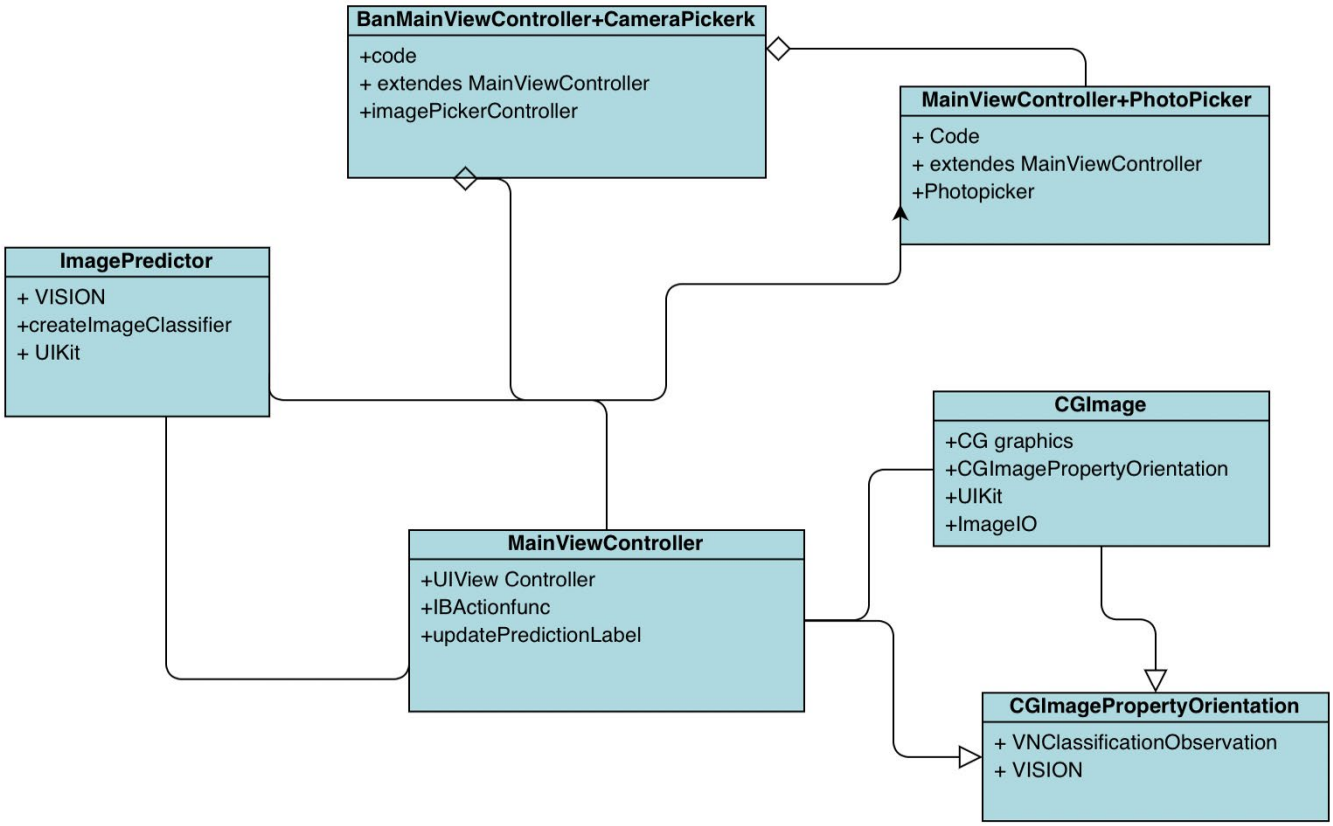


FIGURE -5.1

5.2 Interaction Diagram

5.2.1 Sequence Diagram

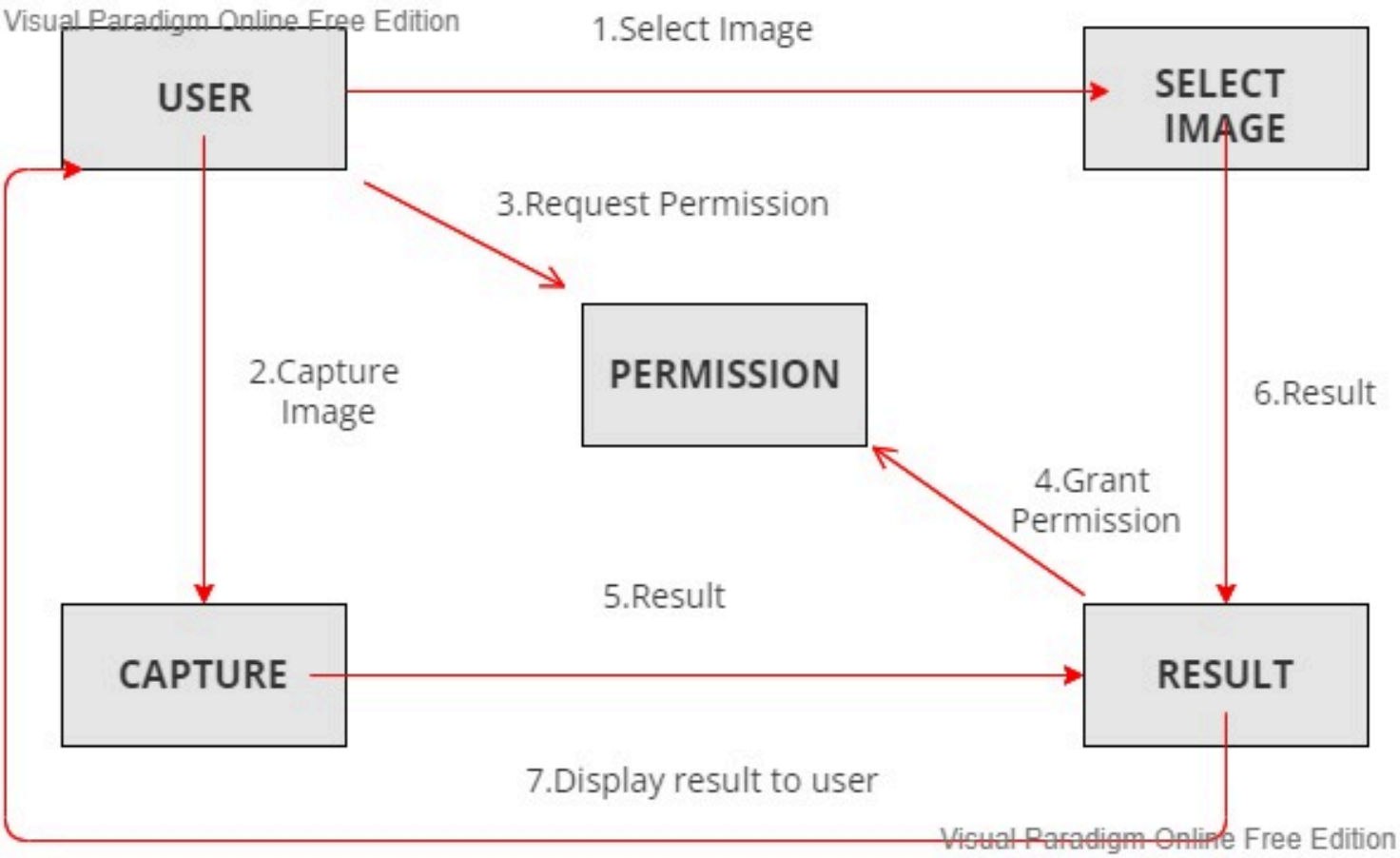


FIGURE -5.2

5.2.2 Collaboration Diagram

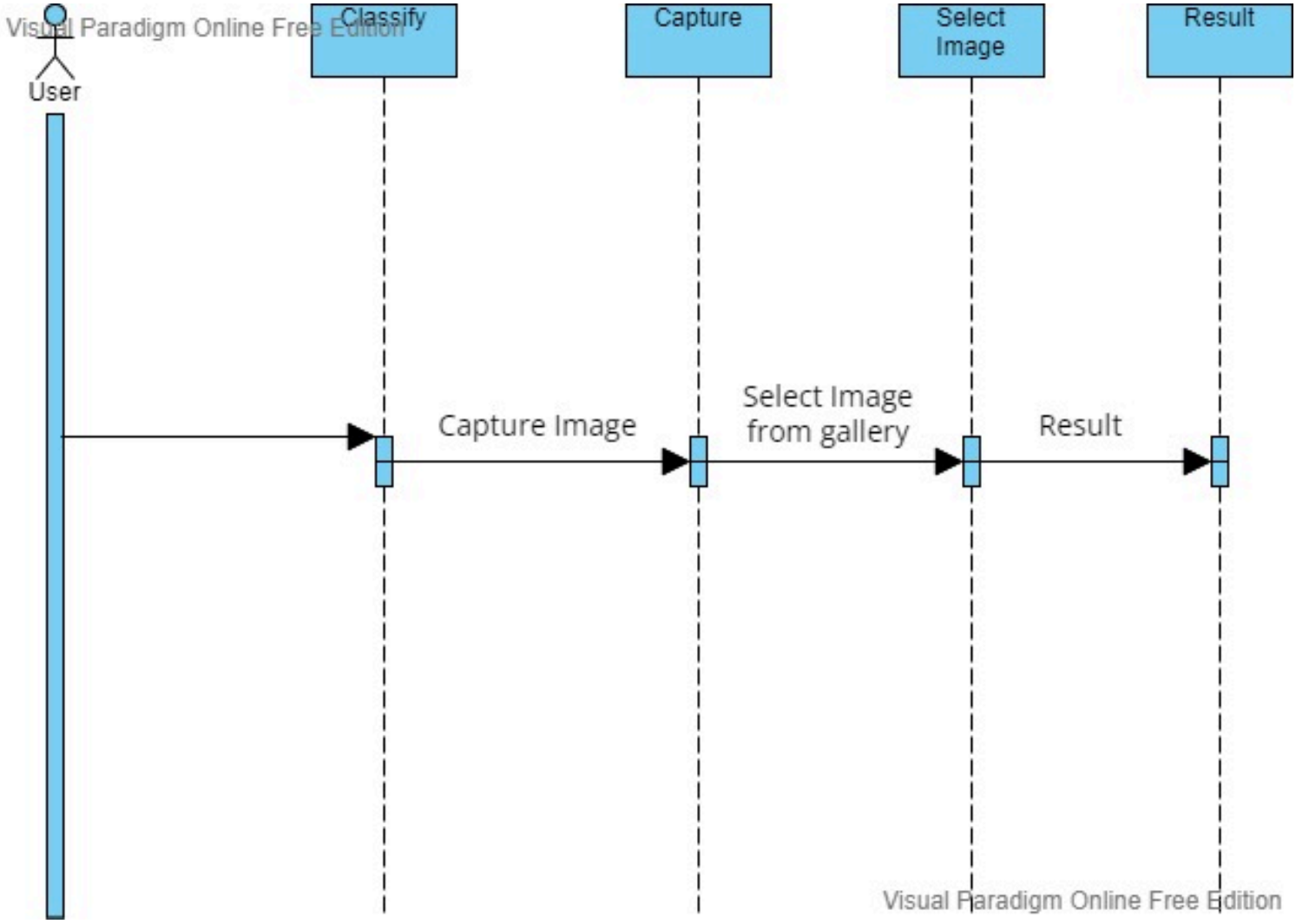


FIGURE -5.3

5.3 State Diagram-

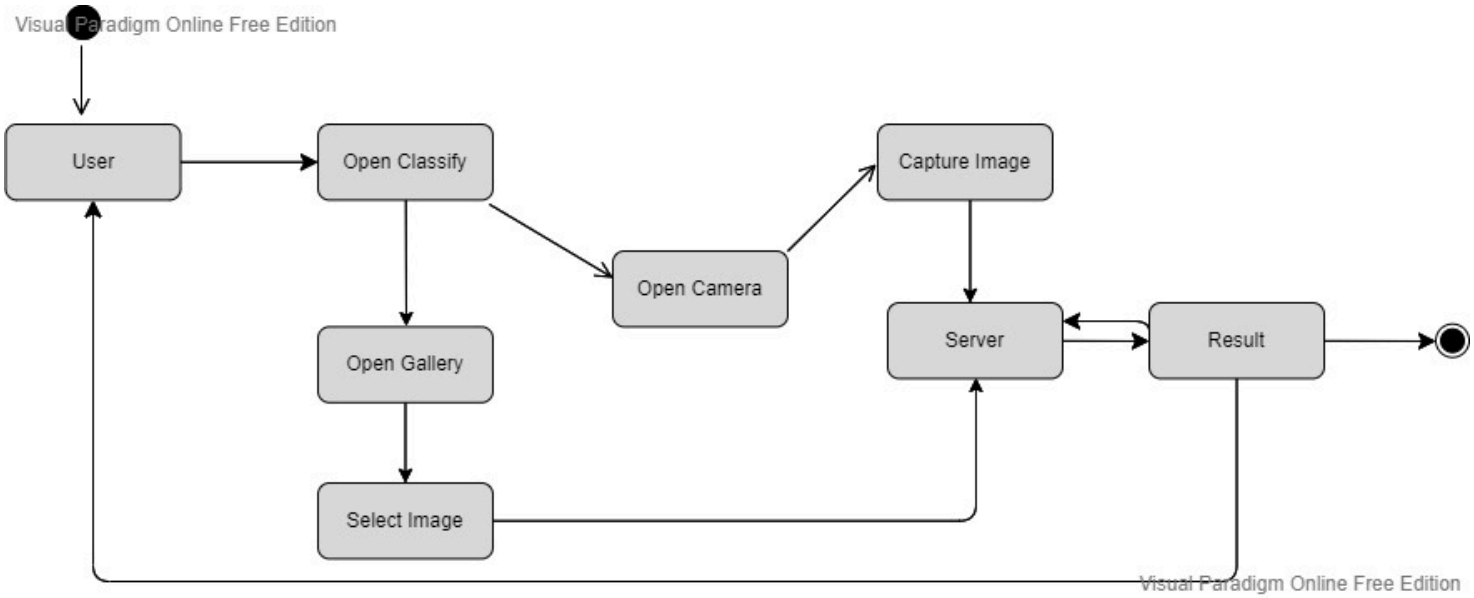


FIGURE -5.4

5.4 Activity Diagram-

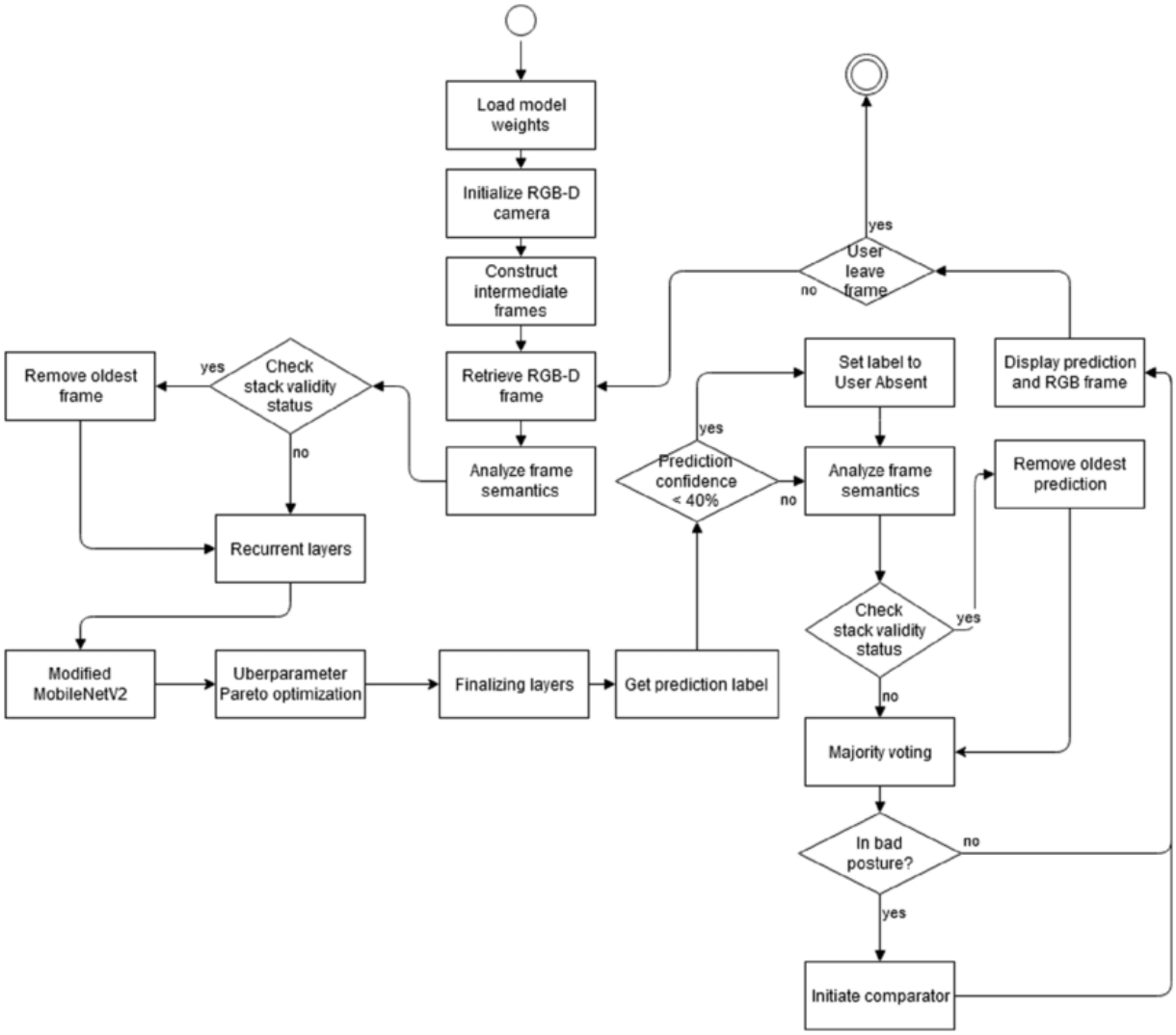
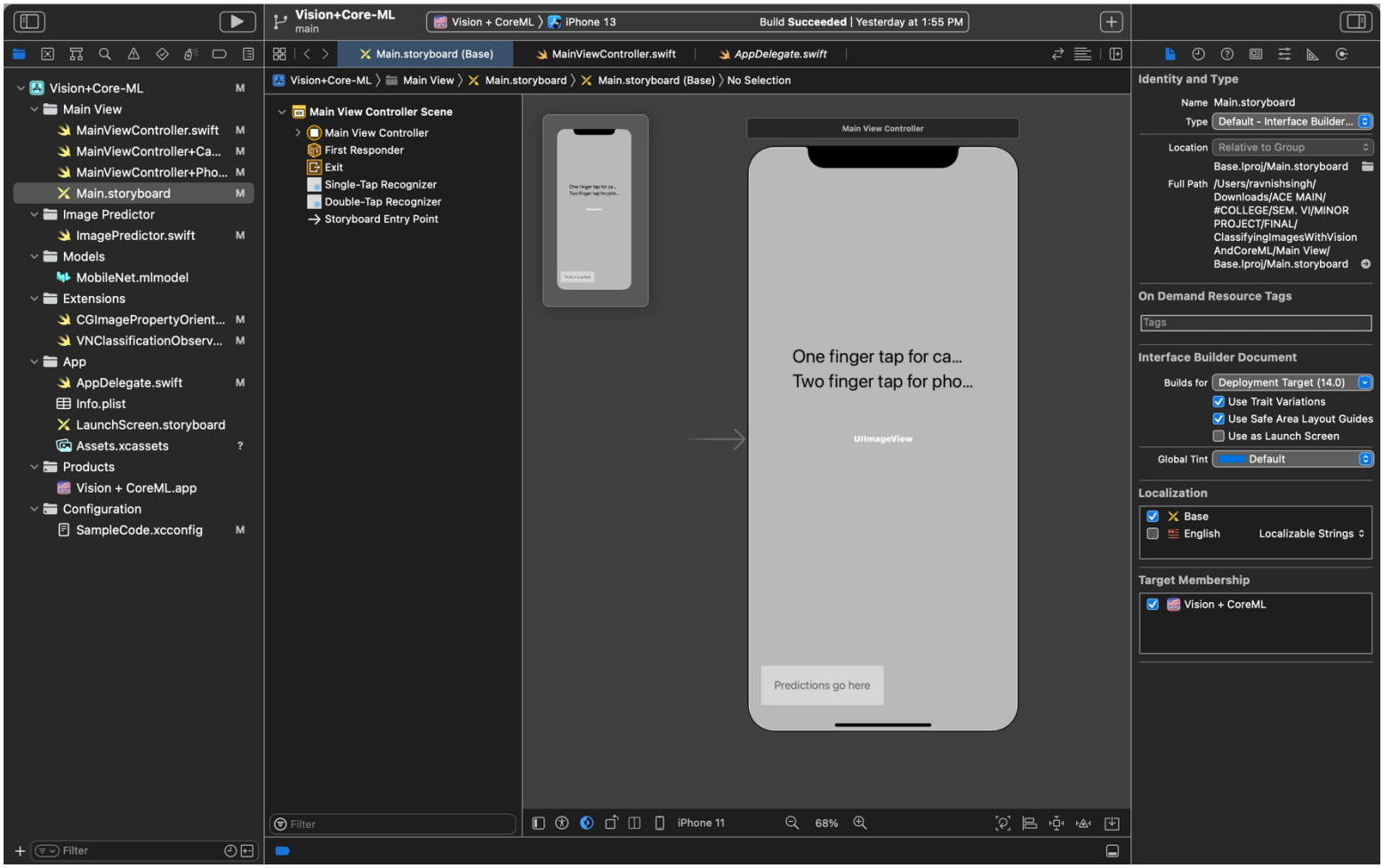


FIGURE -5.5

5.6 Component Diagram-

5.6.1 Deployment Diagram-



5.7 Testing –

The Table-1 shows the different Test Cases.
Test Results –

TC-	INPUT	OUTPUT	RESULT
TC1	When image is chosen as input.	Image with bounding box around the objects and predicted class	SUCCESSFUL
TC2	When video is chosen as input	Video with bounding box around the objects and predicted class	SUCCESSFUL
TC3	When camera is chosen as input	Objects detected in the real time with bounding box, confidence score and predicted class	SUCCESSFUL
TC4	When black and white image is taken as input.	Image with bounding box around the objects and predicted class	SUCCESSFUL
TC5	Image with far objects is taken as input	Image with detected objects	UNSUCCESSFUL
TC6	When image with overlapping objects is	Image with bounding box around the objects and	SUCCESSFUL
TC7	When image with far objects is taken as input	Image with detected objects	UNSUCCESSFUL

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embedding and segmentation similar to how other popular large scale models. MobileNet has **accuracy 65% in 100 epochs**. But as we can see in the training performance of MobileNet, its accuracy is getting improved and it can be inferred that the accuracy will certainly be improved if we run the training for more number of epochs.

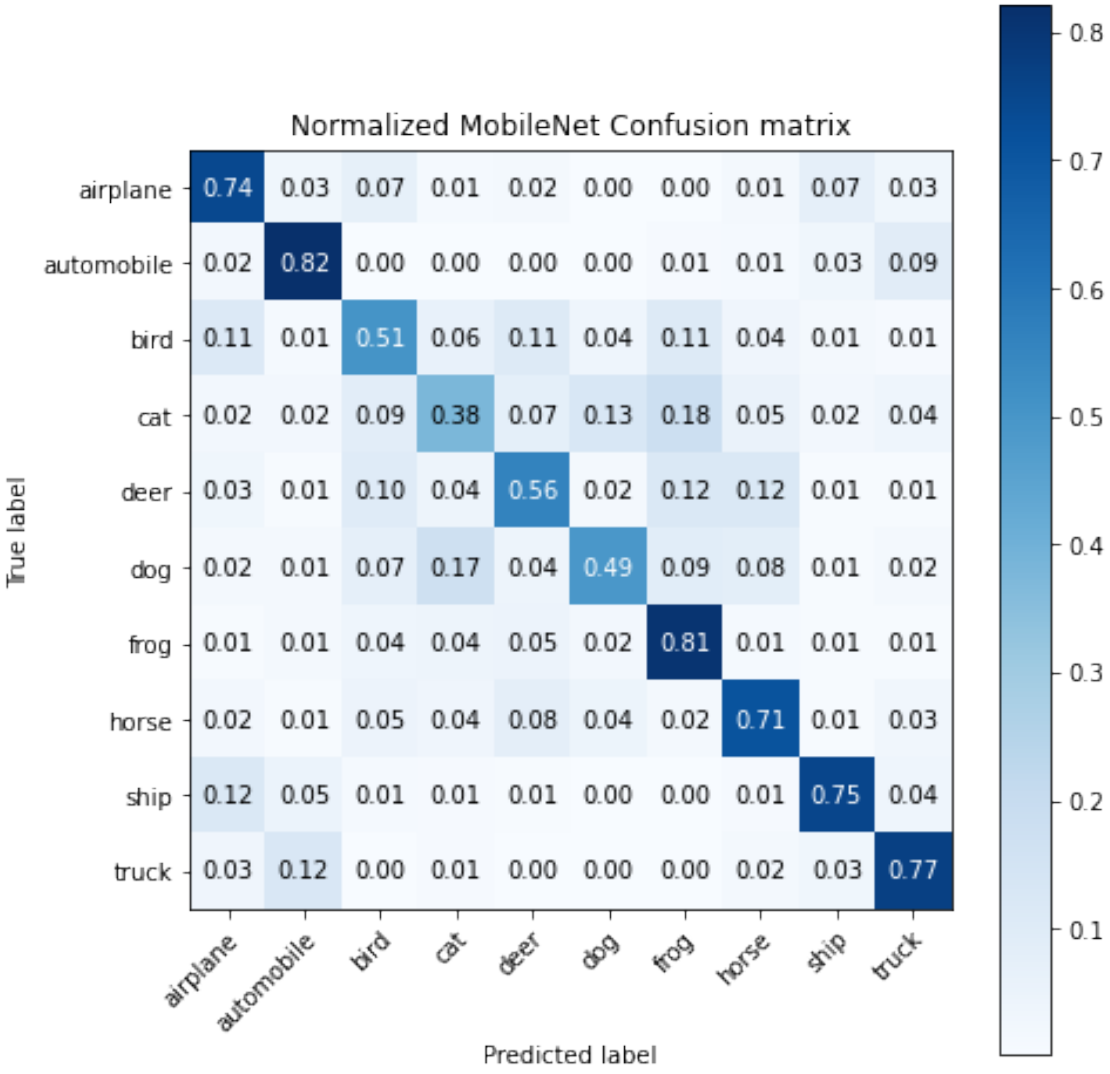


FIGURE -5.6

CHAPTER-6
CONCLUSION & FUTURE WORK

6.1 Limitation of Project-

The only limitation this app contains is that it is only available for IOS Operating system. The project is developed with objective of detecting real time objects in image, video and camera. Bounding Boxes are drawn around the detected objects along with the label indicating the class to which the object belongs. We have used CPU for the processing in the project.

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings, and segmentation, similar to how other popular large scale models, such as Inception, are used.

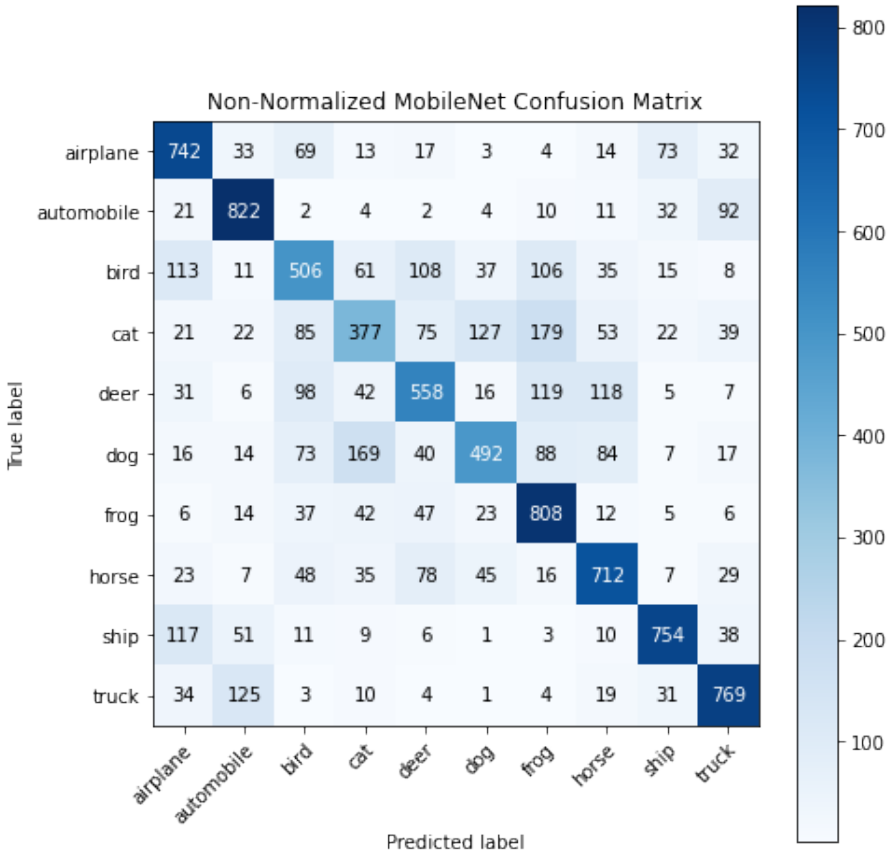


FIGURE -6.1

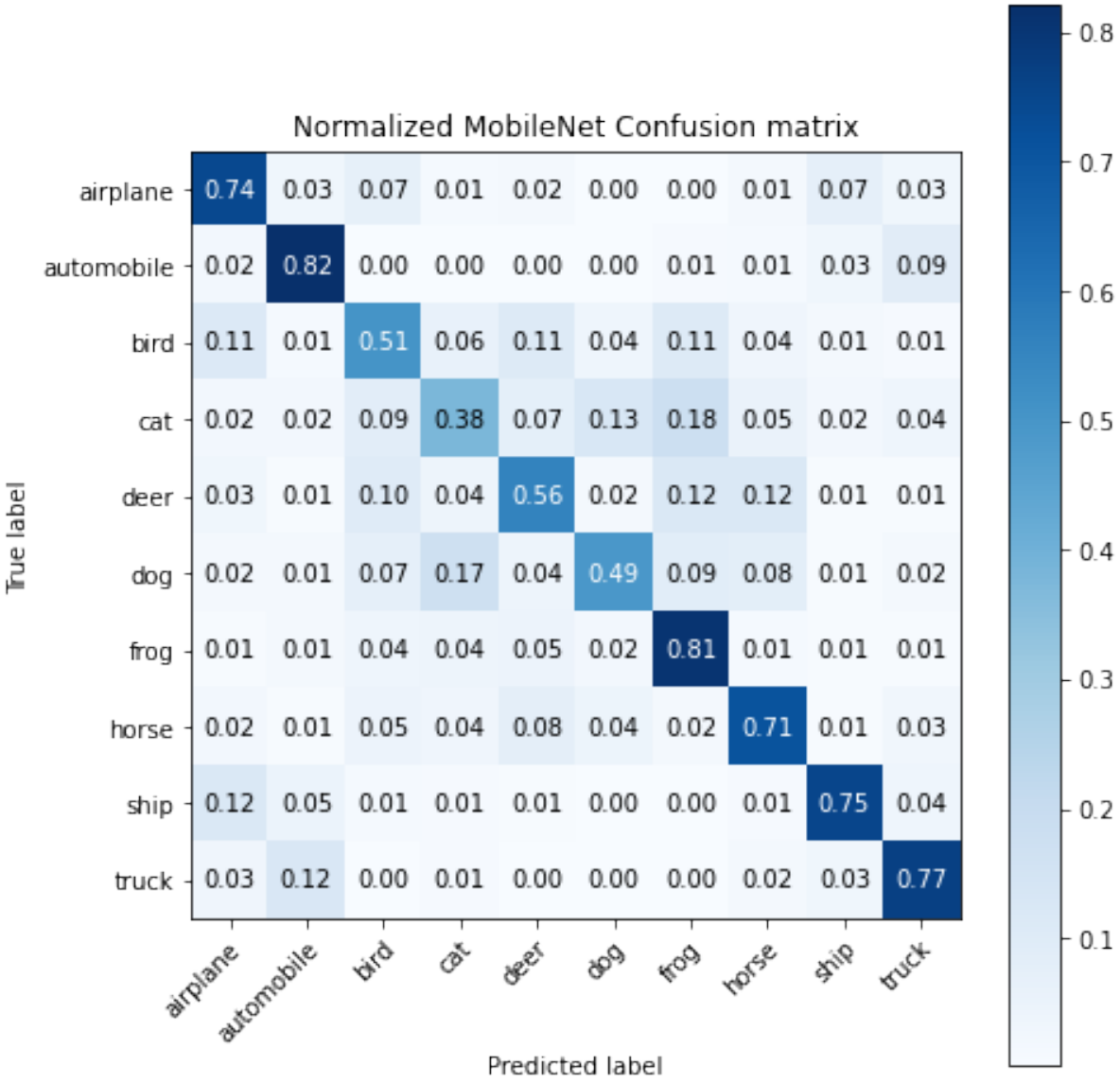


FIGURE -6.2

MobileNet has accuracy 65% in 100 epochs. But as we can see in the training performance of **MobileNet, its accuracy is getting improved** and it can be inferred that the accuracy will certainly be improved if we run the training for more number of epochs.

6.2 Future Scope of the Project –

Object detection in images and video has received lots of attention in the computer vision and pattern recognition communities over recent years. We have had great progress in the field, processing a single image used to take 20 seconds per image and today it takes less than 20 milliseconds.

For humans and many other animals, visual perception is one of the most important senses; we heavily rely on vision whenever we interact with our environment. In order to pick up a glass, we need to first determine which part of our visual impression corresponds to the glass before we can find out where we have to move our hands in order to grasp it

The same code that can be used to recognize Stop signs or pedestrians in a self-driving vehicle signs can also be used to find cancer cells in a tissue biopsy.

If we want to recognize another human, we first have to find out which part of the image we see represents that individual, as well as any distinguishing factors of their face.

Notably, we generally do not actively consider these basic steps, but these steps pose a major challenge for artificial systems dealing with image processing.

Future enhancements can be focused by implementing the project on the system having GPU for faster results and better accuracy.

To add Other Important CoreML like object motion tracking, AI Visual Enhancement, AR, VR.

Indeed, object detection is a key task for most computer and robot vision systems.

Although there has been great progress in the last several years, there will be even bigger improvements in the future with the advent of artificial intelligence in conjunction with existing techniques that are now part of many consumer electronics or have been integrated in assistant driving technologies.

Although the possibilities are endless when it comes to future use cases for object detection, there are still significant challenges remaining.

Herewith are some of the main useful applications of object detection: Vehicle's Plates recognition, self-driving cars, tracking objects, face recognition, medical imaging, object counting, object extraction from an image or video, person detection.

The future of object detection technology is in the process of proving itself, and much like the original Industrial Revolution, it has the potential to free people from menial jobs that can be done more efficiently and effectively by machines. It will also open new avenues of research and operations that will reap additional benefits in the future.

Thus, these challenges circumvent the need for a lot of training requiring a massive number of datasets to serve more nuanced tasks, with its continued evolution, along with the devices and techniques that make it possible, it could soon become the next big thing in the future.

CHAPTER-7
APPENDIX

7.1 Classes-

Training a Create ML Model to Classify Flowers
Train a flower classifier using Create ML in Swift Playgrounds, and apply the resulting model to real-time image classification using Vision.

class VNCoreMLRequest
An image analysis request that uses a Core ML model to process images.

class VNClassificationObservation
An object that represents classification information that an image analysis request produces.

class VNPixelBufferObservation
An object that represents an image that an image analysis request produces.

class VNCoreMLFeatureValueObservation
An object that represents a collection of key-value information that a Core ML image analysis request produces.

Create an Image Classifier Instance-

At launch, the ImagePredictor class creates an image classifier singleton by calling its createImageClassifier() type method.

```
/// - Tag: namestatic func createImageClassifier() -> VNCoreMLModel {
 // Use a default model configuration.

 let defaultConfig = MLModelConfiguration()
    // Create an instance of the image classifier's wrapper class.

  let imageClassifierWrapper = try? MobileNet(configuration: defaultConfig)

  guard let imageClassifier = imageClassifierWrapper else {       fatalError("App failed to create
an image classifier model instance.")    }

    // Get the underlying model instance.

let imageClassifierModel = imageClassifier.model
    // Create a Vision instance using the image classifier's model instance.
guard let imageClassifierVisionModel = try?

VNCoreMLModel(for: imageClassifierModel)
else {        fatalError("App failed to create a `VNCoreMLModel` instance.")    }

    return imageClassifierVisionModel}
```

The method creates a Core ML model instance for Vision by:

1. Creating an instance of the model's wrapper class that Xcode auto-generates at compile time
2. Retrieving the wrapper class instance's underlying MLModel property
3. Passing the model instance to a VNCoreMLModel initializer

The Image Predictor class minimizes runtime by only creating a single instance it shares across the app.

Note:
Share a single VNCoreMLModel instance for each Core ML model in your project.

Create an Image Classification Request-

The Image Predictor class creates an image classification request — a VNCoreMLRequestinstance — by passing the shared image classifier model instance and a request handler to its initializer.

```
// Create an image classification request with an image classifier model.

let imageClassificationRequest =
VNCoreMLRequest(model: ImagePredictor.imageClassifier,
completionHandler: visionRequestHandler)
imageClassificationRequest.imageCropAndScaleOption = .centerCrop
```

The method tells Vision how to adjust images that don't meet the model's image input constraints by setting the request's imageCropAndScaleOption property to VNImageCropAndScaleOption.centerCrop.

Create a Request Handler-

The Image Predictor's makePredictions(for photo, ...) method creates a VNImageRequestHandler for each image by passing the image and its orientation to the initializer.

```
let handler = VNImageRequestHandler(cgImage: photoImage, orientation: orientation)
```

Vision rotates the image based on orientation — a CGImagePropertyOrientationinstance — before sending the image to the model.
If the image you want to classify has a URL, create a Vision image request handler with one of these initializers:

- init(url:options:)
- init(url:orientation:options:)

Start the Request-

The makePredictions(for photo, ...) method starts the request by adding it into a VNRequest array and passes it to the handler's perform(_:) method.

let requests: [VNRequest] = [imageClassificationRequest]

// Start the image classification request.try handler.perform(requests)

Note:
You can perform multiple Vision requests on the same image by adding each request to the array you pass to the perform(_:) method's requests parameter.

Retrieve the Request's Results

When the image classification request is finished, Vision notifies the Image Predictor by calling the request's completion handler, visionRequestHandler(_:error:). The method retrieves the request's results by:

1. Checking the error parameter
2. Casting results to a VNClassificationObservation array

// Cast the request's results as an `VNClassificationObservation` array.guard
let observations = request.results as? [VNClassificationObservation]
else {
// Image classifiers, like MobileNet, only produce classification observations.
// However, other Core ML model types can produce other observations.
// For example, a style transfer model produces `VNPixelBufferObservation` instances.

print("VNRequest produced the wrong result type: \(type(of: request.results)).") return}

// Create a prediction array from the observations.predictions = observations.map { observation in

// Convert each observation into an `ImagePredictor.Prediction` instance.

Prediction(classification:observation.identifier, confidencePercentage: observation.confidencePercentageString)}

The Image Predictor converts each result to Prediction instances, a simple structure with two string properties.

The method sends the predictions array to the Image Predictor's client — the main view controller — by calling the client's completion handler.

// Send the predictions back to the

client.predictionHandler(predictions)

Format and Present the Predictions-

The main view controller's imagePredictionHandler(_:) method formats the individual predictions into a single string and updates a label in the app's UI using helper methods.

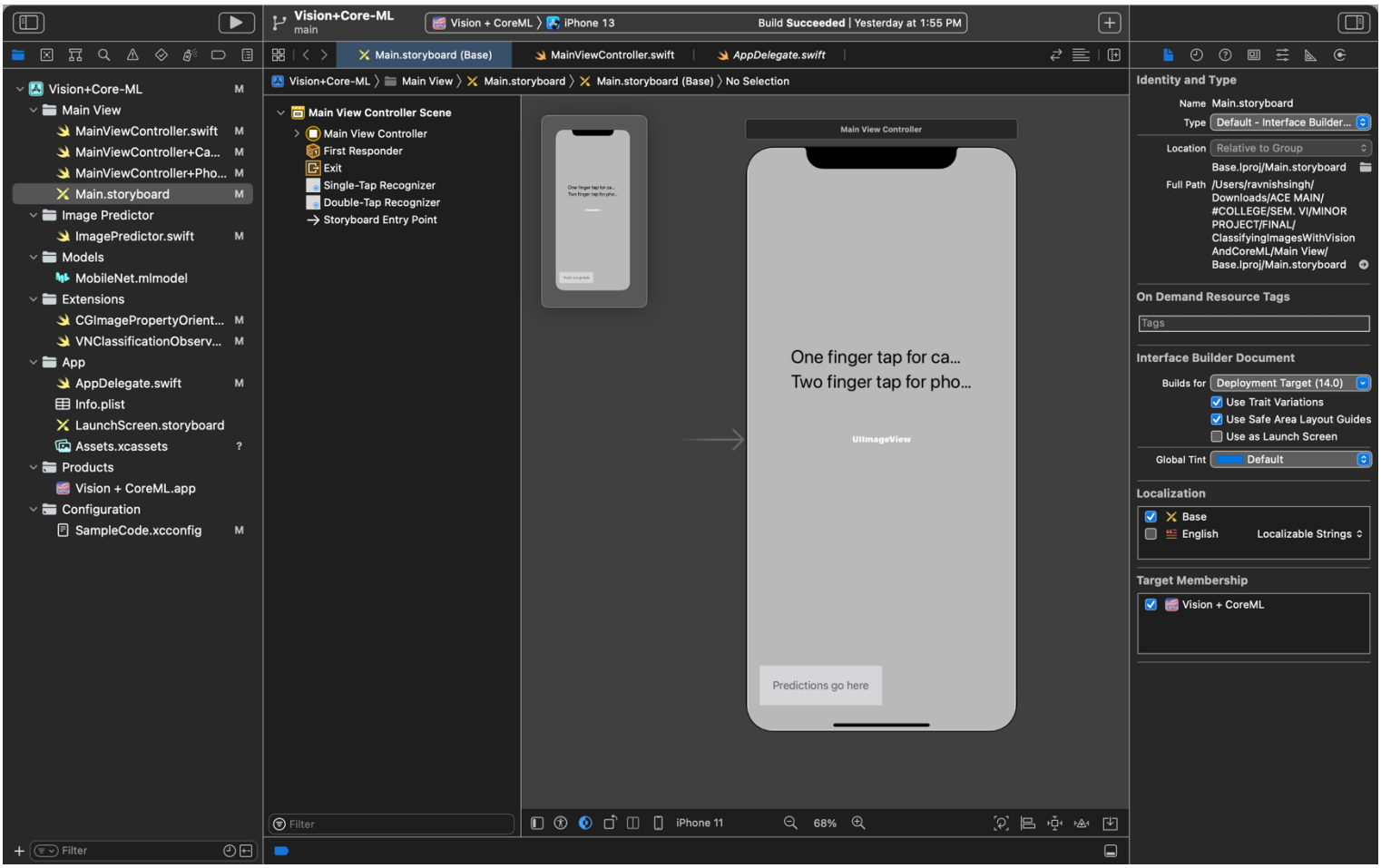
```
private func imagePredictionHandler(_ predictions: [ImagePredictor.Prediction]?) {

   guard let predictions = predictions
else {
  updatePredictionLabel("No predictions. (Check console log.)")

return    }

let formattedPredictions = formatPredictions(predictions)

let predictionString = formattedPredictions.joined(separator: "\n")
  updatePredictionLabel(predictionString)
}
```

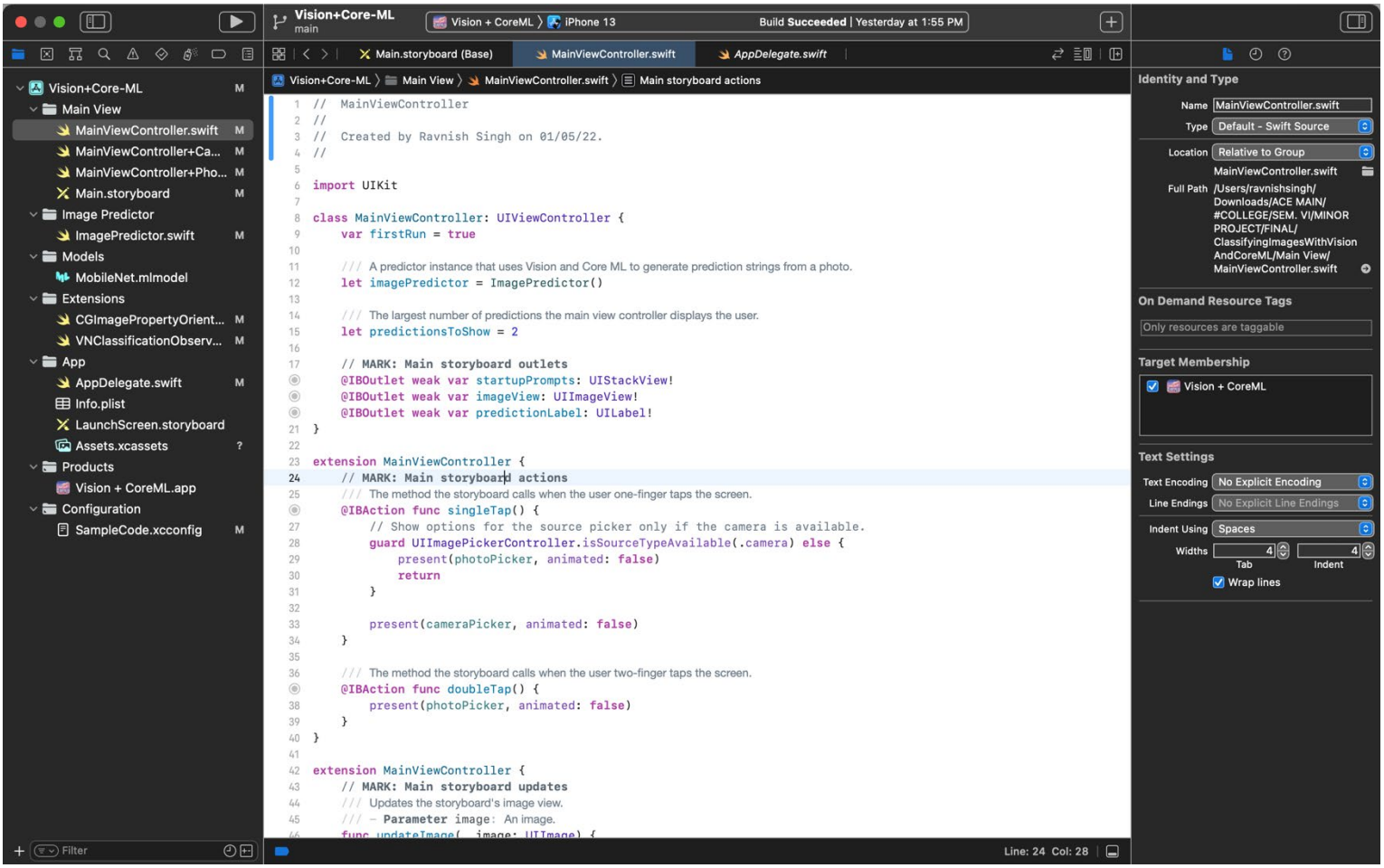
The updatePredictionLabel(_:) helper method safely updates the UI by updating the label's text on the main dispatch queue.

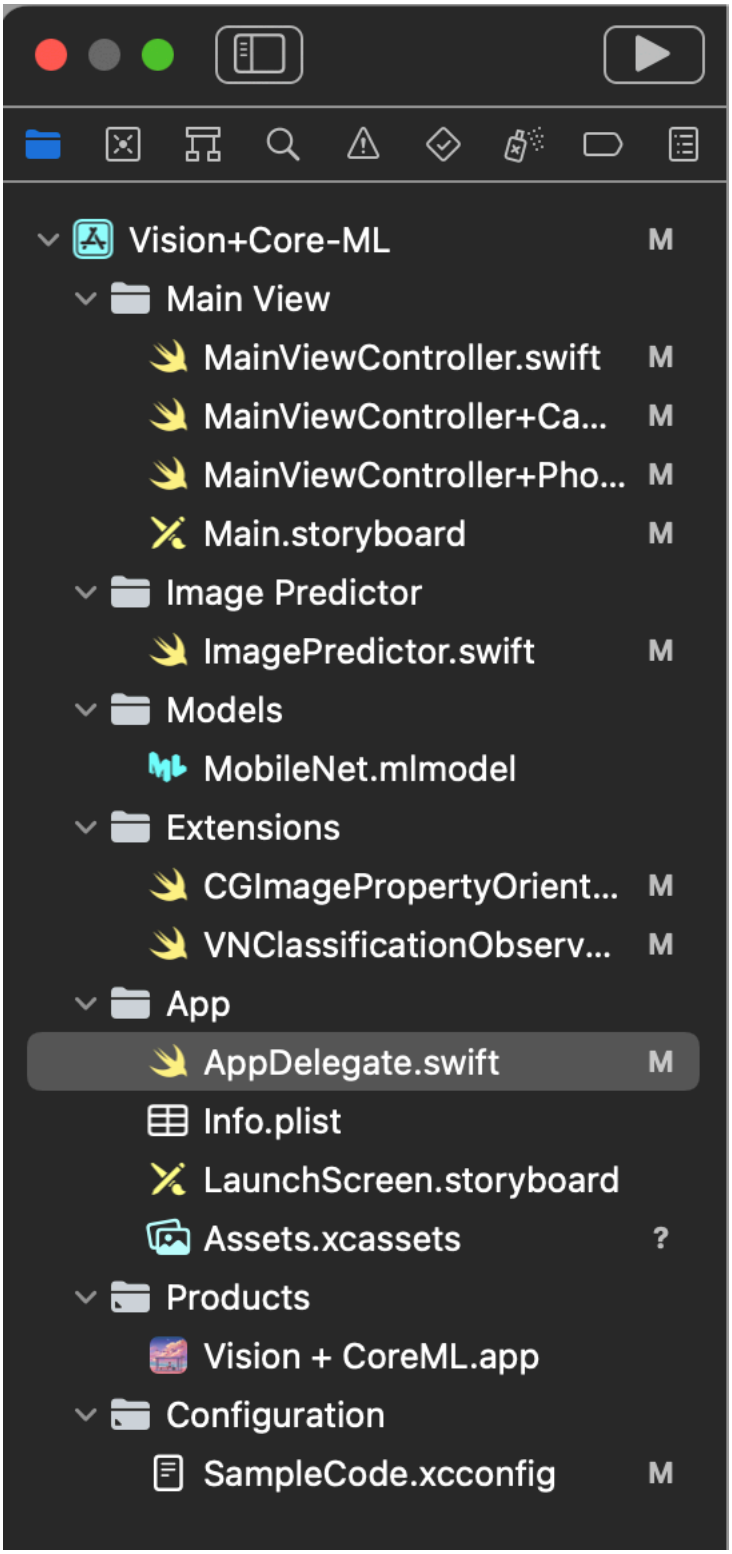
```
func updatePredictionLabel(_ message: String)
{
  DispatchQueue.main.async {
       self.predictionLabel.text = message
}
```

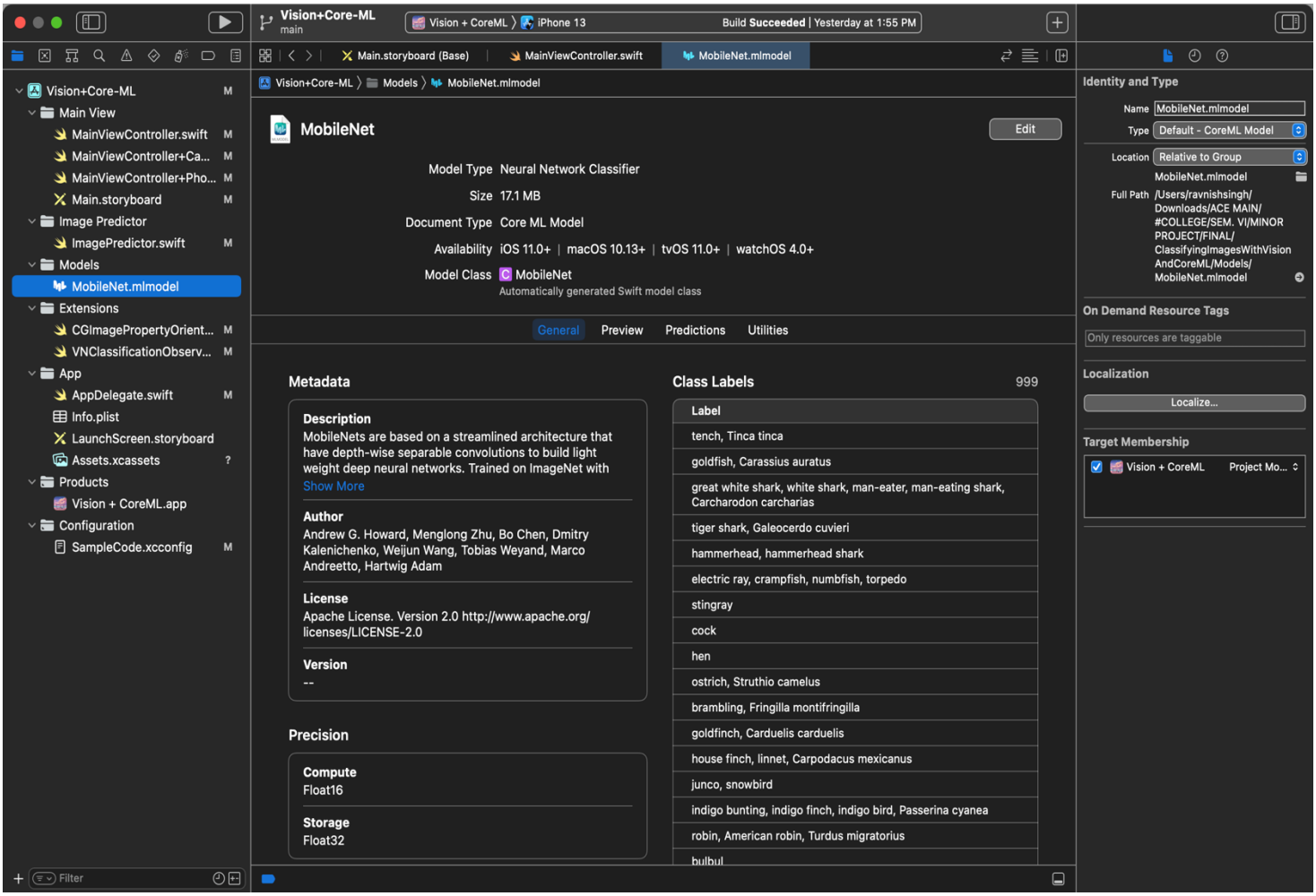
Important
Keep your app's UI responsive by making predictions with Core ML models off of the main thread.

7.2 Snapshot-


Vision+Core-ML
main
Vision + CoreML 〉 iPhone 13
Build Succeeded | Yesterday at 1:55 PM
Main.storyboard (Base)
MainViewController.swift
AppDelegate.swift
Vision+Core-ML 〉 Main View 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 No Selection
Vision+Core-ML
Main View
MainViewController.swift
MainViewController+Ca...
MainViewController+Pho...
Main.storyboard
Image Predictor
ImagePredictor.swift
Models
MobileNet.mlmodel
Extensions
CGImagePropertyOrient...
VNClassificationObserv...
App
AppDelegate.swift
Info.plist
LaunchScreen.storyboard
Assets.xcassets
Products
Vision + CoreML.app
Configuration
SampleCode.xcconfig
Main View Controller Scene
Main View Controller
First Responder
Exit
Single-Tap Recognizer
Double-Tap Recognizer
Storyboard Entry Point
Main View Controller
One finger tap for ca...
Two finger tap for pho...
UIImageView
Predictions go here
iPhone 11
68%
Identity and Type
Name Main.storyboard
Type Default - Interface Builder...
Location Relative to Group
Base.lproj/Main.storyboard
Full Path /Users/ravnishsingh/Downloads/ACE MAIN/#COLLEGE/SEM. VI/MINOR PROJECT/FINAL/ClassifyingImagesWithVisionAndCoreML/Main View/Base.lproj/Main.storyboard
On Demand Resource Tags
Tags
Interface Builder Document
Builds for Deployment Target (14.0)
Use Trait Variations
Use Safe Area Layout Guides
Use as Launch Screen
Global Tint Default
Localization
Base
English
Localizable Strings
Target Membership
Vision + CoreML


Vision+Core-ML
main
Vision + CoreML 〉 iPhone 13
Build Succeeded | Yesterday at 1:55 PM
Main.storyboard (Base)
MainViewController.swift
AppDelegate.swift
Vision+Core-ML 〉 Main View 〉 MainViewController.swift 〉 Main storyboard actions
Vision+Core-ML
Main View
MainViewController.swift
MainViewController+Ca...
MainViewController+Pho...
Main.storyboard
Image Predictor
ImagePredictor.swift
Models
MobileNet.mlmodel
Extensions
CGImagePropertyOrient...
VNClassificationObserv...
App
AppDelegate.swift
Info.plist
LaunchScreen.storyboard
Assets.xcassets
Products
Vision + CoreML.app
Configuration
SampleCode.xcconfig
Filter
// MainViewController
//
// Created by Ravnish Singh on 01/05/22.
//
import UIKit
class MainViewController: UIViewController {
var firstRun = true
/// A predictor instance that uses Vision and Core ML to generate prediction strings from a photo.
let imagePredictor = ImagePredictor()
/// The largest number of predictions the main view controller displays the user.
let predictionsToShow = 2
// MARK: Main storyboard outlets
@IBOutlet weak var startupPrompts: UIStackView!
@IBOutlet weak var imageView: UIImageView!
@IBOutlet weak var predictionLabel: UILabel!
}
extension MainViewController {
// MARK: Main storyboard actions
/// The method the storyboard calls when the user one-finger taps the screen.
@IBAction func singleTap() {
// Show options for the source picker only if the camera is available.
guard UIImagePickerController.isSourceTypeAvailable(.camera) else {
present(photoPicker, animated: false)
return
}
present(cameraPicker, animated: false)
}
/// The method the storyboard calls when the user two-finger taps the screen.
@IBAction func doubleTap() {
present(photoPicker, animated: false)
}
}
extension MainViewController {
// MARK: Main storyboard updates
/// Updates the storyboard's image view.
/// - Parameter image: An image.
Line: 24 Col: 28
Identity and Type
Name MainViewController.swift
Type Default - Swift Source
Location Relative to Group
MainViewController.swift
Full Path /Users/ravnishsingh/Downloads/ACE MAIN/#COLLEGE/SEM. VI/MINOR PROJECT/FINAL/ClassifyingImagesWithVisionAndCoreML/Main View/MainViewController.swift
On Demand Resource Tags
Only resources are taggable
Target Membership
Vision + CoreML
Text Settings
Text Encoding No Explicit Encoding
Line Endings No Explicit Line Endings
Indent Using Spaces
Widths 4 4
Tab Indent
Wrap lines


Vision+Core-ML M
Main View
MainViewController.swift M
MainViewController+Ca... M
MainViewController+Pho... M
Main.storyboard M
Image Predictor
ImagePredictor.swift M
Models
MobileNet.mlmodel
Extensions
CGImagePropertyOrient... M
VNClassificationObserv... M
App
AppDelegate.swift M
Info.plist
LaunchScreen.storyboard
Assets.xcassets ?
Products
Vision + CoreML.app
Configuration
SampleCode.xcconfig M


Vision+Core-ML
main
Vision + CoreML 〉 iPhone 13
Build Succeeded | Yesterday at 1:55 PM
Main.storyboard (Base)
MainViewController.swift
MobileNet.mlmodel
Vision+Core-ML 〉 Models 〉 MobileNet.mlmodel
Vision+Core-ML
Main View
MainViewController.swift
MainViewController+Ca...
MainViewController+Pho...
Main.storyboard
Image Predictor
ImagePredictor.swift
Models
MobileNet.mlmodel
Extensions
CGImagePropertyOrient...
VNClassificationObserv...
App
AppDelegate.swift
Info.plist
LaunchScreen.storyboard
Assets.xcassets
Products
Vision + CoreML.app
Configuration
SampleCode.xcconfig
Filter
MobileNet
Edit
Model Type Neural Network Classifier
Size 17.1 MB
Document Type Core ML Model
Availability iOS 11.0+ | macOS 10.13+ | tvOS 11.0+ | watchOS 4.0+
Model Class MobileNet
Automatically generated Swift model class
General
Preview
Predictions
Utilities
Metadata
Description
MobileNets are based on a streamlined architecture that have depth-wise separable convolutions to build light weight deep neural networks. Trained on ImageNet with
Show More
Author
Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam
License
Apache License. Version 2.0 http://www.apache.org/licenses/LICENSE-2.0
Version
--
Precision
Compute
Float16
Storage
Float32
Class Labels
999
Label
tench, Tinca tinca
goldfish, Carassius auratus
great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias
tiger shark, Galeocerdo cuvieri
hammerhead, hammerhead shark
electric ray, crampfish, numbfish, torpedo
stingray
cock
hen
ostrich, Struthio camelus
brambling, Fringilla montifringilla
goldfinch, Carduelis carduelis
house finch, linnet, Carpodacus mexicanus
junco, snowbird
indigo bunting, indigo finch, indigo bird, Passerina cyanea
robin, American robin, Turdus migratorius
bulbul
Identity and Type
Name MobileNet.mlmodel
Type Default - CoreML Model
Location Relative to Group
MobileNet.mlmodel
Full Path /Users/ravnishsingh/Downloads/ACE MAIN/#COLLEGE/SEM. VI/MINOR PROJECT/FINAL/ClassifyingImagesWithVisionAndCoreML/Models/MobileNet.mlmodel
On Demand Resource Tags
Only resources are taggable
Localization
Localize...
Target Membership
Vision + CoreML
Project Mo...

CHAPTER-8
BIBLIOGRAPHY & REFERENCES

7.1 Reference Books, links, and docs -

7.1.1 BOOKS-

- The Swift Programming Language Edition - 5.3.
- App Development with Swift.

7.1.2 LINKS-

- https://developer.apple.com/documentation/vision/classifying_images_with_vision_and_core_ml.
- https://developer.apple.com/documentation/vision.
- https://developer.apple.com/documentation/coreml.
- https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470.