OWAIS QAYYUM IOS MOBILE APPLICATION FOR FOOD AND LOCATION IMAGE PREDICTION USING CNN NEU 2018

IOS MOBILE APPLICATION FOR FOOD AND LOCATION IMAGE PREDICTION USING CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES OF NEAR EAST UNIVERSITY

By
OWAIS QAYYUM

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Software Engineering

NICOSIA, 2018

IOS MOBILE APPLICATION FOR FOOD AND LOCATION IMAGE PREDICTION USING CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES OF NEAR EAST UNIVERSITY

By
OWAIS QAYYUM

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Software Engineering

NICOSIA, 2018

Owais QAYYUM: IOS MOBILE APPLICATION FOR FOOD AND LOCATION IMAGE PREDICTION USING CONVOLUTIONAL NEURAL NETWORKS

Approval of Director of Graduate School of Applied Sciences

Prof. Dr. Nadire Çavuş

We certify this thesis is satisfactory for the award of the degree of Master of Science in Software Engineering

Examining Committee in Charge:

Assoc. Prof. Dr. Kamil Dimililer	Head of Department, Automotive Engineering, NEU
Asst. Prof Dr. Yoney Kirsal Ever	Head of Department, Software Engineering, NEU
Assoc. Prof Dr. Melike Şah Direkoglu	Supervisor, Department of Computer Engineering, NEU

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name:

Signature:

Date:

To my parents

ACKNOWLEDGEMENTS

First of all, I highly appreciate the unconditional support of my supervisor Assoc. Prof. Dr. Melike Şah Direkoglu throughout the whole research study. For sure, without her support, motivation, and guidance, this thesis would have never been possible. I also pass my gratitude towards my department for allowing me to carry out the research study and all their assistance that each member provided.

ABSTRACT

Machine Learning is a popular research area in software industry alongside with big data, micro services, virtual reality, and augmented reality. With the recent developments in improving computing capacity, deep learning approaches such as Convolutional Neural Networks (CNN) has become the trendiest topic in machine learning for image recognition. In this paper, we have developed an IOS application for food image recognition using modified CNN models. In particular, we developed an IOS mobile application by converting the machine learning models to CoreML and then using them within the IOS application for food image recognition and location prediction. We used Python as a programming language to training the CNN model while Anaconda as an IDE. While for converting the machine learning to CoreML we used CAFFE framework and CoreML tools. For IOS mobile application we used Xcode as an IDE and Swift as a programming language. In this research we used supervised learning as we got the food dataset from food 101 which is already labelled. We used RGB images over greyscale images because greyscale images have 256 combinations of shades of grey per pixel while RGB has 16,777,216 colour combinations per pixel. The colour image is input to the convolutional neural network for automatic feature extraction and training. We took our CNN model to be inception V3 model as its top-5 error rate was very low. Seven layers are used in Inception V3 model and also its computation time and cost are very low comparing to other CNN models. We trained our model on MacBook pro 2017 having 8GB ram and 2.4Ghz core i5 processor. Our mobile Application can be easily downloaded by users from the Apple App store. In our research we discussed that the classification and prediction time vary between different IOS mobile devices. Image classification and prediction takes more time on iPhone 6 and less time on iPhone X. Comparing to other machine learning models we trained our model in just 6 hours with 100,000 images and got astonishing results. Our prediction results show that we can achieve an accuracy of 82% Top-1, 87% Top-3 and 97.00% Top-5 for our food prediction model.

***Keywords*:** convolutional neural networks; machine learning; deep learning; image recognition; CoreML; TensorFlow; Keras; Python; Data Mining.

ÖZET

Makine Öğrenimi büyük veri, mikro hizmetler, sanal gerçeklik ve artırılmış gerçeklik ile birlikte yazılım endüstrisinde popüler bir araştırma alanıdır. Bilgi işlem kapasitesinin geliştirilmesindeki son gelişmelerle birlikte, Konvolüsyonel Sinir Ağları (CNN) gibi derin öğrenme yaklaşımları, görüntü tanıma için makine öğrenmesinde en popüler konu haline geldi. Bu yazıda, değiştirilmiş CNN modelleri kullanılarak gıda görüntü tanıma için bir IOS uygulaması geliştirdik. Özellikle, makine öğrenme modellerini CoreML'ye dönüştürerek ve ardından bunları gıda görüntüsü tanıma ve konum tahmini için IOS uygulaması içinde kullanarak bir IOS mobil uygulaması geliştirdik. Python'u CNN modelini, Anaconda'yı IDE olarak eğitmek için programlama dili olarak kullandık. Makine öğrenmesini CoreML'ye dönüştürürken CAFFE framework ve CoreML araçlarını kullandık. IOS mobil uygulaması için Xcode'u IDE, Swift ise programlama dili olarak kullandık. Bu araştırmada denetimli öğrenmeyi kullandık, çünkü zaten etiketli olan gıda 101'den alınan veri veri setini aldık. Gri tonlamalı resimler üzerinde RGB görüntüleri kullandık, çünkü gri tonlamalı görüntüler piksel başına 256 gri renk kombinasyonuna sahipken, RGB piksel başına 16.777.216 renk kombinasyonuna sahip. Renkli görüntü, otomatik özellik çıkarma ve eğitim için evrişimli sinir ağına girilir. İlk 5 hata oranı çok düşük olduğu için CNN modelimizi başlangıç V3 modeline aldık. Inception V3 modelinde yedi katman kullanılmıştır ve hesaplama süresi ve maliyeti diğer CNN modellerine kıyasla çok düşüktür. Modelimizi 8GB ram ve 2.4Ghz core i5 işlemcili MacBook pro 2017'de eğittik. Mobil Uygulamamız, kullanıcılar tarafından Apple App Store'dan kolayca indirilebilir. Araştırmamızda, sınıflandırma ve tahmin süresinin farklı IOS mobil cihazları arasında değiştiğini tartıştık. Görüntü sınıflandırma ve öngörme, iPhone 6'da daha fazla, iPhone X'te daha az zaman alır. Diğer makine öğrenme modelleriyle karşılaştırıldığında, modelimizi sadece 6 saatte 100.000 görüntüyle yetiştirdik ve şaşırtıcı sonuçlar aldık. Tahmin sonuçlarımız, gıda tahmin modelimiz için% 82 Top-1,% 87 Top-3 ve% 97.00 Top-5 doğruluğunu elde edebileceğimizi gösteriyor.

Anahtar Kelimeler: konvolüsyonel sinir ağları; makine öğrenimi; derin öğrenme; görüntü tanıma, CoreML; TensorFlow; Keras; Python; Veri Madenciliği.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

IDE	Integrated Development Environment
ML	Machine Learning
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
NLP	Natural Language Processing
API	Application Programming Interface
AI	Artificial Intelligence

CHAPTER 1
INTRODUCTION

Most people know about the term “Machine learning” and “Neural networks” but they do not know it’s real meaning. Machine learning has different algorithms among which neural networks has become very popular in the software industry. Existence of neural networks can easily be acknowledged in most of the digital services and is also known as a recommended system. Let’s take an example, Spotify is a music-based application that offers “Your daily mixes”, or “Recommended stations”, also “Recommend” section in one of the most famous video search engine YouTube, and also “Inspired by your shopping” from Amazon which is using the big data from the daily behaviour of customers. That big data is then analysed using neural network algorithm.

Neural network (Sharma, 2018) has been a modern saga for the public and even for the developers as well. Because of the neural network’s complexity, we developed a convolutional neural network and shown that how it can be useful for public in daily life. With the evaluation of python and open source libraries such as TensorFlow (Hope, 2017) and Keras, machine learning applications (Müller, 2016) are getting easier to build. With the help of IDEs such as Anaconda, Hydrogen lab and JupyterLab, it became easier to implement machine learning techniques. Advanced neural network architectures have evolved rapidly to promote the use of machine learning.

1.1 Theoretical background

We have noticed that the terms Machine Learning, Artificial Intelligence, Deep Learning (Suskie, 2001) and Neural Networks are found to be same in different documents and articles. In most of the cases reader thinks that all of them are same. Figure 1.1 demonstrate some of the differences between Deep Learning, Machine Learning and Artificial Intelligence (Institute of Electrical and Electronics Engineers, 2013).

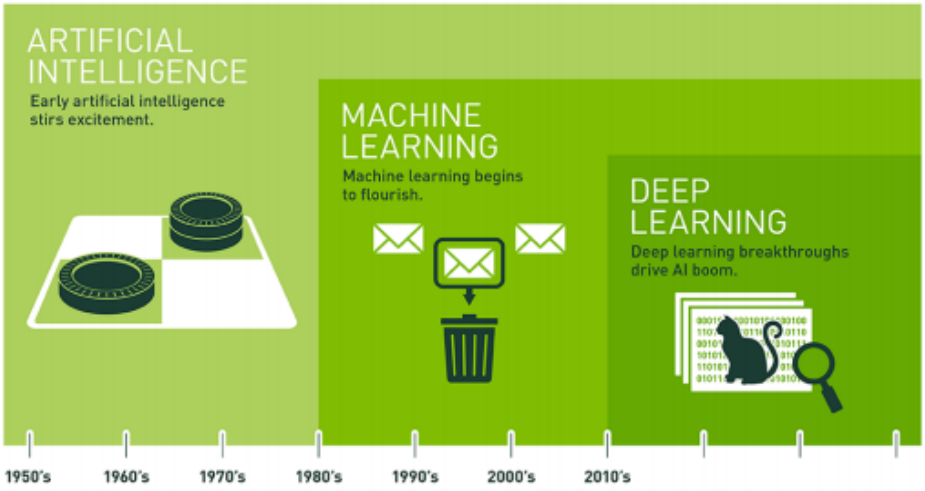


Figure 1.1 Machine Learning, Artificial Intelligence and Deep Learning connections

As we can see from Figure 1.1 Artificial intelligence is here since 1950 where machine learning is the subset or a part of artificial intelligence which began to flourish since 1980. If we talk about deep learning then it's the subset of Machine learning which came into existence in 2010 and its breakthroughs drive Artificial Intelligence boom.

1.1.1 Machine learning

Machine learning is a field of computer science which started with the beginning of computer science history. Alan Turing, in 1950 (Turing & Yang, 2013), founder of computer science, asked the question "Can machines think?" which set the very first milestone for machine learning studies. Later Arthur Samuel defined machine learning as "field of study that gives computers the ability to learn without being explicitly programmed". However, machine learning was finally defined by Tom M. Mitchell:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with the experience E."

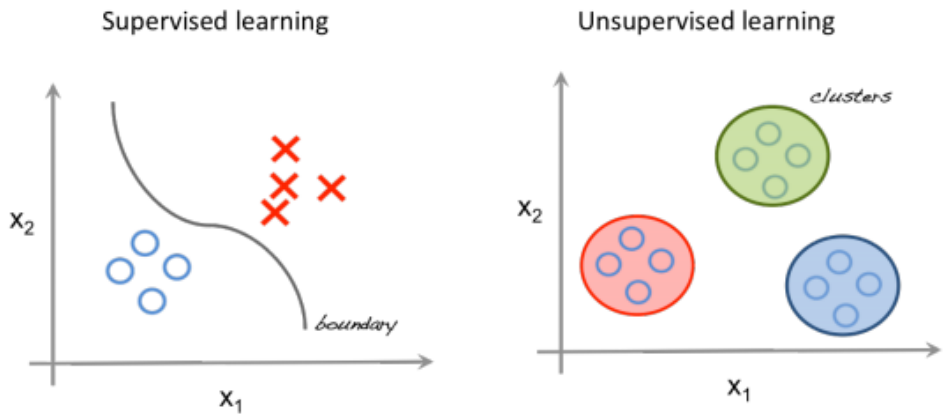


Figure 1.2 Supervised vs Unsupervised learning

1.1.1.1 Supervised Learning

Machine learning can be divided into three major types among them supervised learning is mostly used. Results are clearly categorized with clear conditions in supervised learning (Liu, Datta, & Lim, 2014). For example: labelling fruits name corresponding to its features, predicting if a person is male or female, predicting stock exchange price, etc.

1.1.1.2 Unsupervised Learning

Second type of machine learning is Unsupervised Learning. In this type of machine learning, results are clearly categorized with abstract conditions and non-labelled. For example: grouping students with different IQ levels, categorizing different types of oil used for lubricating vehicles, etc.

1.1.2 Convolutional Neural Networks

Convolutional Neural Network (Venkatesan, 2018) or Conv Neural Network also known as CNN, a technique within the broader Deep Learning field, have been a revolutionary force in Computer Vision applications especially in the past half-decade. One main use-case is that of image classification, e.g. determining whether a picture is that of a dog or cat.

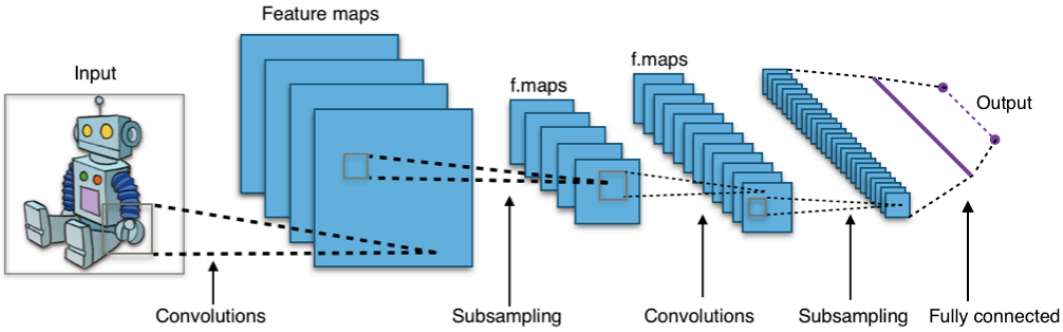


Figure 1.3 Convolution Neural Networking

You don't have to limit yourself to a binary classifier of course; CNNs can easily scale to thousands of different classes, as seen in the well-known ImageNet dataset of 1000 classes, used to benchmark computer vision algorithm performance.

In the past couple of years, these cutting-edge techniques have started to become available to the broader software development community. Industrial strength packages such as TensorFlow have given us the same building blocks that Google uses to write deep learning applications for embedded/mobile devices to scalable clusters in the cloud -- Without having to hand code the GPU matrix operations, partial derivative gradients, and stochastic optimizers that make efficient applications possible. On top of all of this, are user-friendly APIs such as Keras that abstract away some of the lower level details and allow us to focus on rapidly prototyping a deep learning computation graph. Much like we would mix and match Legos to get a desired result.

1.1.3 CoreML framework

The CoreML framework is the first basic machine learning framework that was resealed by Apple in 2017. CoreML framework mainly focuses on Natural Language Processing (NLP) and image analysis. Right now, CoreML is having a smaller number of APIs and right now is only available for IOS 11 and above. The general Machine learning application structure is shown in Figure 1.4.

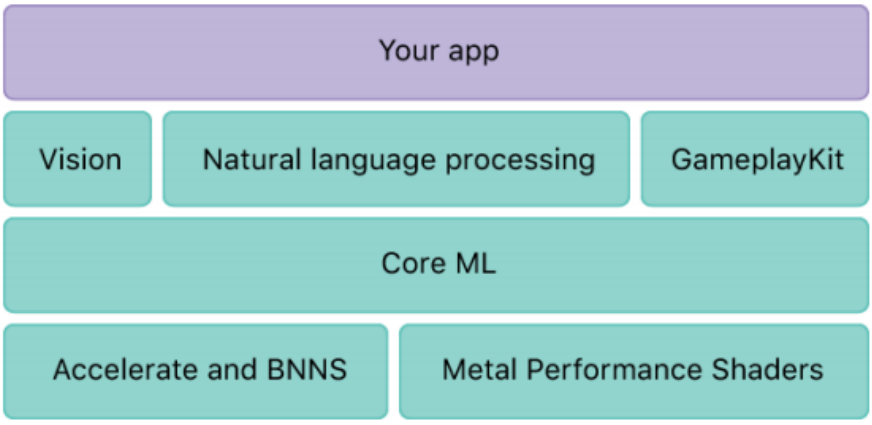


Figure 1.4 Machine Learning based IOS application general structure

From Figure 1.4, CoreML is a middle layer framework which is built on top of machine learning performance primitive frameworks such as Metal Performance Shaders, Accelerate and BNNS (Basic neural network subroutines). After collecting the data from these framework, raw data is then optimised which is processed from the above-mentioned frameworks and send them to higher level frameworks such as NLP, Vision and or Game PlayKit.

CoreML is a powerful framework for machine learning that can be used directly. To integrate the coreml model into a machine learning based IOS application (Keur, 2016), one only need to add a core ML model file to the project. The framework generates the model class that contains all the utilities that can predict the output based on the selected input.

1.2 Aims and Objectives

In this thesis I have developed a novel IOS application based on two machine learning models. For the thesis I have used two machine learning models.

i. Food names dataset which is consist of almost 5000 images.
ii. Places images from Sultan Ankara Mosque to Washington's White house covering almost 4 million places.

We have created a new model for food prediction and reused a pre-built model for location prediction. These models were later converted to CoreML and integrated into IOS mobile application. We took our CNN model to be inception V3 model as its top-5 error rate was very low. When we compared the Inception V3 model with models such as AlexNet, Inception (GoogleNet) and BN-Inception V2 model. Results show that we can achieve an accuracy of 97.00% for our food prediction model. So, in particular we developed an IOS mobile application by converting the machine learning models to **CoreML** model and then using them in the IOS application. The aims and objectives of this thesis are as under:

a. One of the purposes of our research was to combine machine learning technologies with IOS mobile application. To achieve this, we took two machine learning models and converted them to CoreML models which are supported by IOS environment Swift.
b. The second purpose is to introduce a technology to travelers which sometimes are not familiar with the name of the food and that's how this application will work. Users just have to take a picture of food and our application will automatically predict the name of the food with a great precision number.
c. The other purpose is to build such an application which will automatically locate the image taken with the camera. Once the users take an image of the surrounding the application will automatically search the dataset of 4 million images and will show 3 best spots on the google maps.
d. Sometimes the users don't have internet availability and they can't use certain applications and can't even surf on internet. Our application doesn't even need any sort of internet connectivity.
e. If you see an image of your friend at a picnic spot so the user can easily find out the location using our application.
f. Users are not bound to select any category or any image. They can use it anytime and even without internet connection.
g. Our app doesn't save the images stored by the users on their device and neither upload them to our database. As the application is on your device and it don't need any internet so it's safe to upload images.

h. As we have used machine technology so the model think itself and give us a precision score.

i. Our plan is to enhance the application and machine learning model in future.

j. The application is on the mobile device and its size is almost 300 megabytes. It’s our future objective to transfer the application to cloud and make its size smaller.

k. It's a novel project as we haven’t seen any IOS application till now that predicts location spots on google maps based on the images provided to the application.

CHAPTER 2
LITERATURE REVIEW

In this chapter, we briefly describe what is Convolutional Neural Network, Machine Learning, Deep Learning, Deep Learning Techniques and App based Prediction.

2.1 Convolution Neural Network

Steve Lawrence in 1997 proposed convolution neural networking approach for face recognition. In this research paper, Steve presented a hybrid neural network which compared with different other methods such as a self-organizing map neural network and CNN system.

Steve took face images which were taken between April 1992 and April 1994 and store them in an ORL database. These images were taken from Olivetti Research Laboratory at Cambridge. His work was mainly based on geometrical features which was proposed by Kanade. Steve also referred to Eigen faces as high-level recognition in which images are been processed in the Marr Paradigm to surface 3D models.

Steve used a system of proceeding parts for face recognition.

i. A typical convolution networks.
ii. System's high-level block diagram.

Various experiments were performed by Steve and presented the results. Steve performed experiments with six testing images and six training images per person for a total of 250 training images and 250 test images with no overlap between the testing and training sets. In this experiment, error rate was 96.5 % at first which is an extremely high error rate.

2.1.1 Saliency Map with Convolution Neural Network

In another research carried out by Seunghoon and Inria (2015) in which they proposed tracking algorithm through CNN. They took a set of pre-learning data through a large image file in offline mode. The algorithm assumes the output of the hidden layers of the network as function descriptors. These functions are also used to learn lens-like discriminative models using an online support vector (SVM) machine. Seunghoon and Irina proposed an algorithm that describes the

complete course of the tracking algorithm. First, they discussed the properties achieved with pre-trained CNN. Then a method of creating a map with a specific highlighting of the destination to show in detail. They also presented an online SVM technique that discriminates and sequentially learns the appearance of the lens.

To evaluate the performance, they use the 50 sequences of the recently published tracking reference dataset (wu et al., 2013). Seunghoon and Inria concluded by suggesting a new visual tracking algorithm based on a pre-trained CNN using the output of CNN's last convolutional layer as generic descriptors of the properties of the objects and the models with discriminating appearance learned online via an SVM line. They used sequential Bayesian filtering with a particular prominence map of the target as an observation.

2.1.2 Deep Inside Convolutional Networks

In another research by Karen and Andrea on April 2014 presented some visualization of image classification models which are learnt using CNN. They consider two visualization techniques which is based on computation of gradient class score.

a. Maximizes the class score by generating the images.
b. Class saliency map computation is done is the second technique which is specific to given image and class.

In their research work, Karen and Andrea also mentioned the research work done by Erhan who visualized deep learning models. They followed the following steps in this research.

a. Class Model Visualization.
b. Image Specific Class Saliency Visualization.
c. Class Saliency Extraction.
d. Weakly Supervised Object Localization.
e. Relation to De-Convolutional Networks.

Karen and Andrea completed their research by introducing two visualization techniques for Conv-Net depth classification. The first technique produces an artificial image representative of a class of interest. The second technology computes a prominence map of the specific image class, illuminating the areas of a given image discriminatively with respect to the given class. Therefore, the highlight map can be used to initialize the object segmentation based on the cutting of graphs without having to train dedicated acquisition or segmentation models.

2.1.3 Learning and Transferring Mid-Level Image Representations Using CNNs

Maxime, Lean and Others in 2012 proposed a research paper in which they demonstrated transferring and learning using CNNs. In their work, they showed how image representations learned with CNNs on large scale. Maxime reused trained layers on the image net dataset.
They discussed some previous related work on transfer learning, visual object classification and deep learning. It's been discussed that the aim of transfer learning is to transfer the knowledge between related source and target domains. Visually the images can be classified using,

i. Histogram Encoding.
ii. Spatial Pooling.
iii. Fisher Vector Encoding.

The techniques used by Maxime and others are as follows:

i. Transferring CNN Weights.
ii. Network Architecture.
iii. Network Training.
iv. Classification.

2.1.4 High Performance Convolutional Neural Networks

Jonathan and Luca from Switzerland developed a deep architecture that published the handwritten digit recognition and best object classification with the error rates of just 2.53%, 19.51%, and 0.42%.

In their research work, Jonathan and Luca evaluated various networks on the hand-written digit benchmark. MNIST and two image classification benchmarks: NORB and CLFAR10. In their

results, they applied NORB and trained the datasets with seven epochs. The error rates on MNIST drop to 1.62%, 0.87% after 2, 7 and 19 epochs.

2.1.5 Image Net Classification with Deep CNN

In another research, Alex and Bya proposed a system where they trained a deep CNN to classify the 1.3 million high resolution images in the image net into 1000 different classes. They got results which were not much good but the way they analyzed the dataset was great. On their test data they got top-1 and top-5 error rates of 39.6% and 18.0%. The neural network they proposed consists of 59 million parameters and 640,000 neurons, consists of five convolutional layers.

They used the Image Net data set, which consists of more than 14 million good resolution tagged images, which belong to some 21,000 image classes. These were taken from the Internet and were tagged by people tagged by humans with the recruitment tool of many Turkish mechanics from Amazon Turk. The architecture of its network included nine layers, four convolutional layers and five fully connected layers.

The neural network architecture that Alex and Bya presented was having 60 Million parameters. Although the 1000 ILSVRC classes cause each training example 10 to impose restriction bits for image-to-label mapping, this is not sufficient to learn so many parameters without significant over-adaptation. The results are summarized as the network achieve top-1 and top-5 test set error rates of 36.4% and 18.1%. According to their results, a large deep CNN on a highly challenging dataset is capable of achieving great results using purely supervised learning.

2.1.6 Very Deep CNN for Large-Scale Image Recognition

Karen and Andrew again in 2015 published a research paper where they proposed a deep convolutional network for basically large-scale image recognition. The main contribution of their work was to increase the depth of a neural network with a very small (3x3) convolutional filters. These findings were on the basis of their Image Net Challenge in March 2015 submission where they secured the top position in the classification and localization and. They also made their two-best performing Convolutional Network models available for public so that researchers can further evaluate the model.

Now to analyze the improvements brought by the increased Convolutional Network depth in fair setting. They used the same designing principles and were inspired by Ciresan et al (2011) and Krizhevsky et al (2012) for configuring their Convolutional Neural Network. The architecture they proposed was to input the fixed size 224x224 RGB image. In their research, they did only one thing and that was the preprocessing in which they subtracted the mean RGB value, computed on the training set from each pixel. The image was passed through a stack of convolutional (conv.) layers where they used filters with a very small receptive field: 3x3.

They followed the Convolutional Network training procedure proposed once by Krizhevsky in 2012. In this work, Karen and Andrew did the following for evaluating a very deep CNN.

a. Evaluating a very deep convolution network
b. Network was having upto 19 layers
c. They used a fixed size 224x224 RGB image.
d. Images were passed through a small receptive field of 3x3.

It has been shown that the depth of the classification accuracy and the current performance in the challenge data set of the image network can be achieved with a conventional Convolutional Network architecture of much greater depth.

2.1.7 Analysis of Previous Work

Table 2.1 Analysis of Previous Work

Title	Dataset	Technology	Error Rate	Application
Food and Location Images Classification Using CNNs **(Our Research)**	101000 Food Images and 4 Million Location Images	Convolution Neural Network, CoreML	2 %	IOS Application
Face Recognition: a CNN Approach	200 Images	Convolution Neural Network	97.5%	No
Online Tracking by Learning Discriminative Saliency Map with CNNs	1 Million Images	Convolution Neural Network	7.5%	No
Deep Inside Convolutional Network: Visualizing image Classification Models and Saliency Map	2 Million Images	Convolution Neural Network	10.5%	No
Learning and Transferring Mid-Level Image Representations Using CNNs	1.2 Million Images	Convolution Neural Network	8.6%	No
High Performance CNNs for Image Classification.	2.5 Million Images	Convolution Neural Network	2.53%, 19.51% and 0.35%	No
Image Net Classification with Deep CNNs	1.2 Million Images	Convolution Neural Network	37.5% and 17.0 %	No
Very Deep CNN for Large-Scale Image Recognition	4 Million Images	Deep Convolution Neural Network	Not Specified	No

2.2. Mobile Applications Used in Machine Learning Techniques

Smartphones or Mobile phones are fast. Device in the lives of people. Implementation of applications Channels like Apple App Store are changing Smart phones in applications phones. Download a variety of applications and It's important to note that today's smartphones are Programmable and with a growing set of powerful and cheap embedded sensors that make it possible. Creation of personal, group and community scales. In our thesis, we explained how we integrated the machine learning model into the IOS application using Xcode.

The reason we choose IOS application is its widely usage and also the method provided by Apple and that is COREML. A trained model is the result of applying an automatic learning algorithm to a training data set. The model meets new input data predictions.

Figure 2.1 Integrate Machine Learning Models into Application

CHAPTER 3
SYSTEM ARCHITECTURE

In this chapter we will briefly discuss various components of the proposed IOS mobile application based on food and location predictor that uses convolution neural network which is also known as ConvNet or CNN that is one of the main class of deep learning or Feed-Forward artificial neural Network (Yang, 2002).

Our food and location predictor application consist of five main parts:

a. Model Building
b. Model testing
c. Model Conversion to CoreML
d. Integration with IOS application
e. User Interface

The feature highlights of the applications are as follows:

a. Offline Image processing (requires no internet connection).
b. Classification with percentage of accuracy.
c. Built in camera and photo selector.
d. System tools: Swift 4 using Xcode 9 and CoreML.

3.1 Food Name Predictor IOS Interface

Choose or capture a food image that you have made yourself, then the application will decide the name of the food automatically with a percentage of accuracy from a dataset of food images.

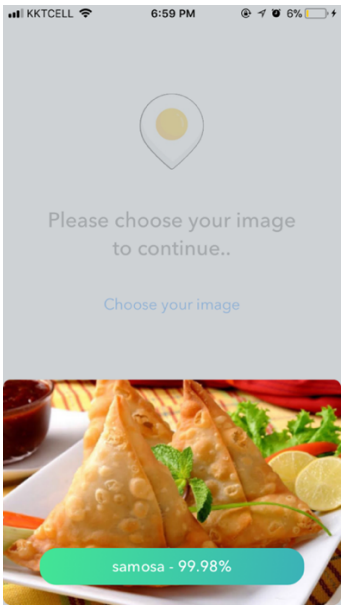


Figure 3.1 Screenshot of IOS application for predicting Samosa image with 99.98% accuracy

3.2 Location Predictor IOS Interface

The Application will predict any image from the dataset of five million images including the world's seven wonders to many different historical and popular places. For any image the application will offer its best three predications.

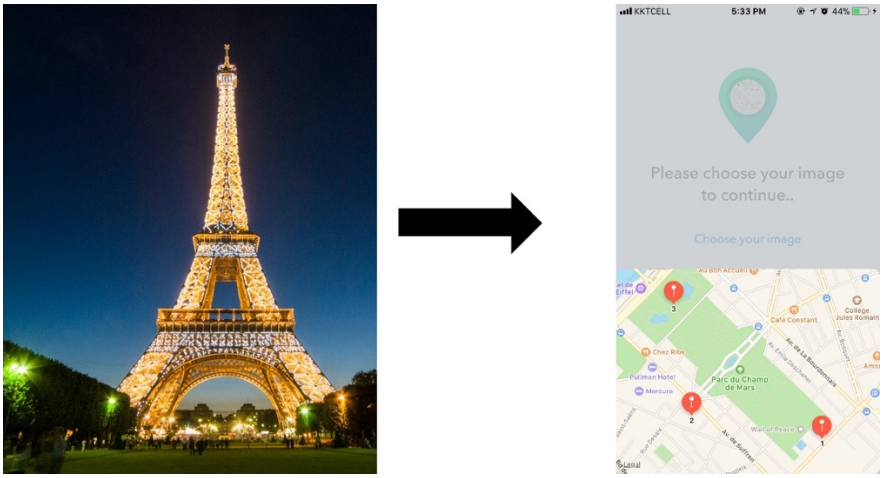


Figure 3.2 An image of Eiffel Tower in Paris was uploaded to the IOS application and its showing three best spots of location on the google maps

3.3 System Overview

All of our application data is stored in the mobile application itself so that there will not be any need of internet connection while using the mobile application. The food dataset is being provided by Food-101 which we took from Kaggle (Ventling, 2012) and Location images machine learning model is used from MIT places database (Zuo, 2016). The food image data set is first converted to a machine learning model and then later on converted to coreML using caffe API. The location model is later converted to coreML using the tools provided by Apple using the Tensorflow API named Keras and Caffe. For this we used the IDE Anaconda. The size of IOS application will be about 300 megabytes as the food prediction do not require any internet and the whole machine learning model will be integrated in the IOS mobile application itself. Internet is only required when predicting locations using google maps.

Deep learning technology helps to extract knowledge from different types of data such as audio, images, videos and texts. In this project we have used image-based recognition. Figure 3.3 illustrates the complete overview of the system.

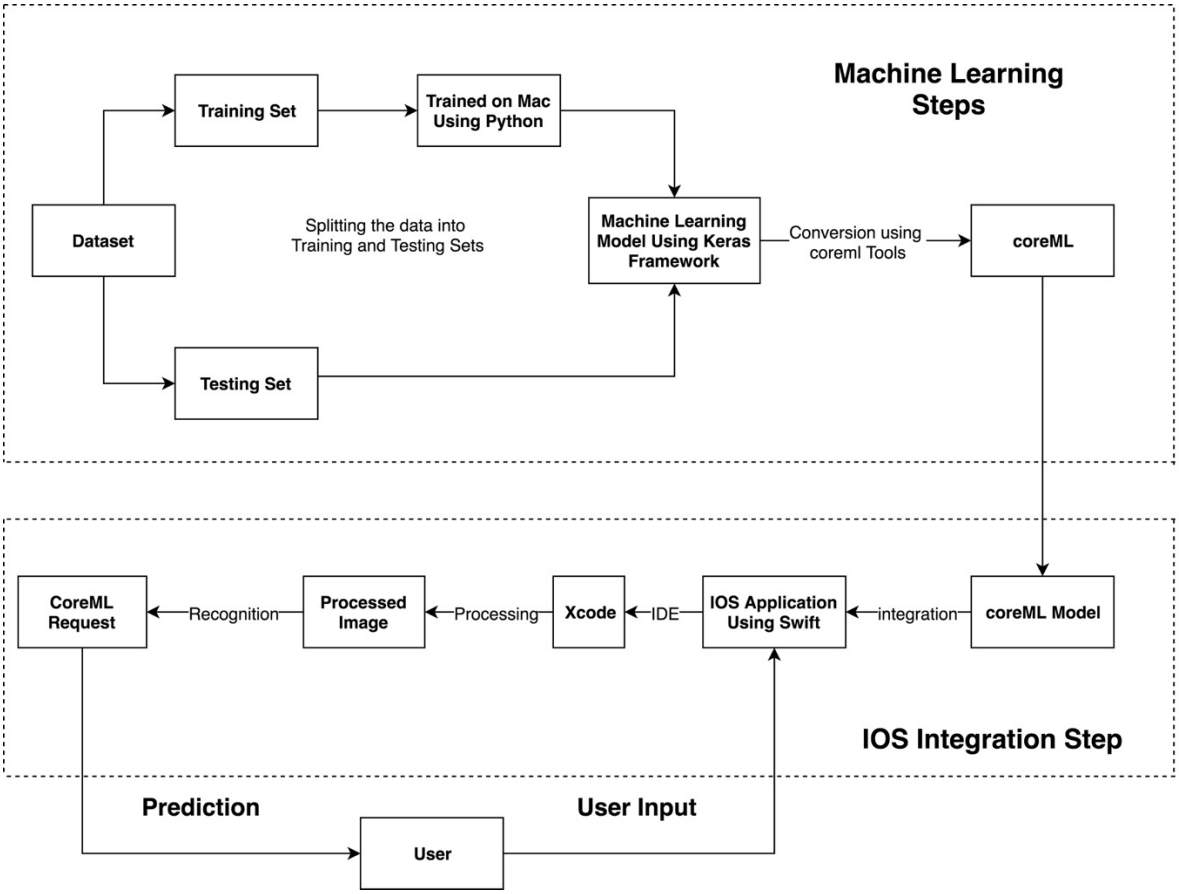


Figure 3.3 System architecture of the CNN model

In figure 3.3, the whole mechanism is divided into two parts:

a) Machine Learning model creation
b) Integration of Model into IOS Application

Deep learning techniques consist of many classes. The two common learning algorithms among them are supervised learning (Convolutional Neural Networks) and unsupervised learning (Restricted Boltzmann Machines). In our system, we mainly focused on supervised learning algorithms which means that the provided dataset has labels. First of all, the dataset was split into training and testing sets. In this case, the training set was 75% while testing set was 25%. Machine learning framework was then applied on the datasets and a model was generated. First of all, dataset is prepared which involves data parsing, indexing the variables and splitting the dataset into testing and training datasets. Different layers were then added such as Convolution2D layer, MaxPooling2D, Dense Layer, Pooling Layers, Dropout Layers, Softmax Layer and Activation Layer. After that the model was fitted and compiled through finding the loss functions and metrics the model is finally evaluated and prediction of data was done. Here a question arises that why we used keras? For superfast implementation and good extensibility of implementing our idea and for doing a deep research in deep learning.

The model was then converted to coreML model using the Tensorflow API named keras. For this conversion the Anaconda was used as an IDE [3]. Now a question arises here that why we need to convert the model specifically to coreML? Conversion to coreML allows developers to train machine learning models within Xcode using Swift and MacOS Playgrounds. Since it is an IOS mobile application that's why we used coreML to be integrated. As Xcode only accept machine learning models that are converting into Xcode standards and that's coreML. In figure 3.4, the complete architecture of conversion to coreML model is been shown.

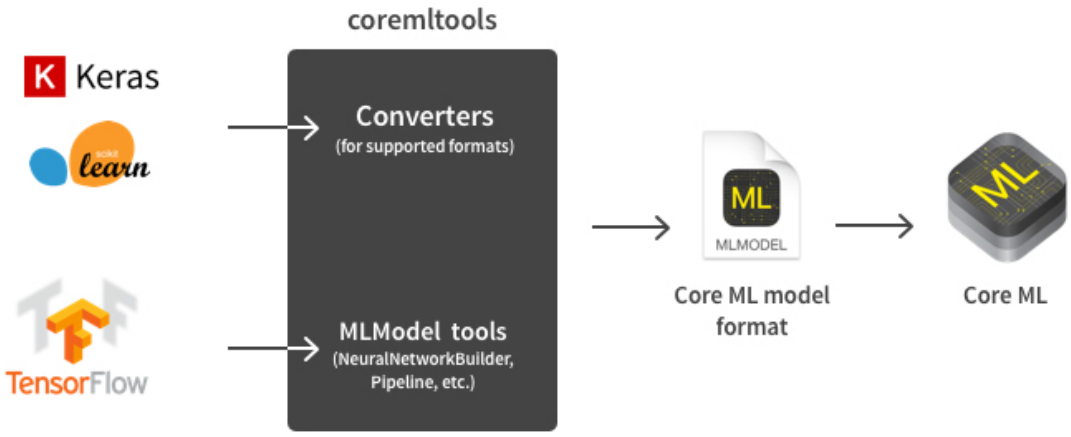


Figure 3.4 Conversion tools from machine learning model to coreML

Once the coreML model is generated, it is now suitable for integration with the Xcode. We have used Xcode as we want to convert a machine learning model to CoreMl and to be tested on IOS mobile phone. As Xcode is the IDE for developing Iphone or Macbook based applications that's why we used Xcode. After the integration, the application is designed and coded in Swift and the features are arranged such that when the user opens the camera or selects an image, it will be processed and results will be displayed in the form of percentage to the user. Before 2014 the standard programing language for developing IOS applications was Objective C. Apple in 2014 announced swift programming language to be officially used for IOS apps development. Here is a snippet of Swift code for our prediction based IOS mobile application.

```
extension String {

var banana : String {

                let shortName = self.dropFirst()

                return "\(self) \(self) Bo B\(shortName) Banana Fana Fo
F\(shortName)"}}

let bananaName = "Jimmy".banana
```

Figure 3.5 Swift Code Snippet

3.4 Used Supervised Learning Approach for Food Image Prediction:

Our approach uses supervised learning algorithms for prediction of food images. Now a days, it is feasible to demonstrate big scale supervised learning using CNN with the help of well annotated dataset like ImageNet (Tang, 2018).

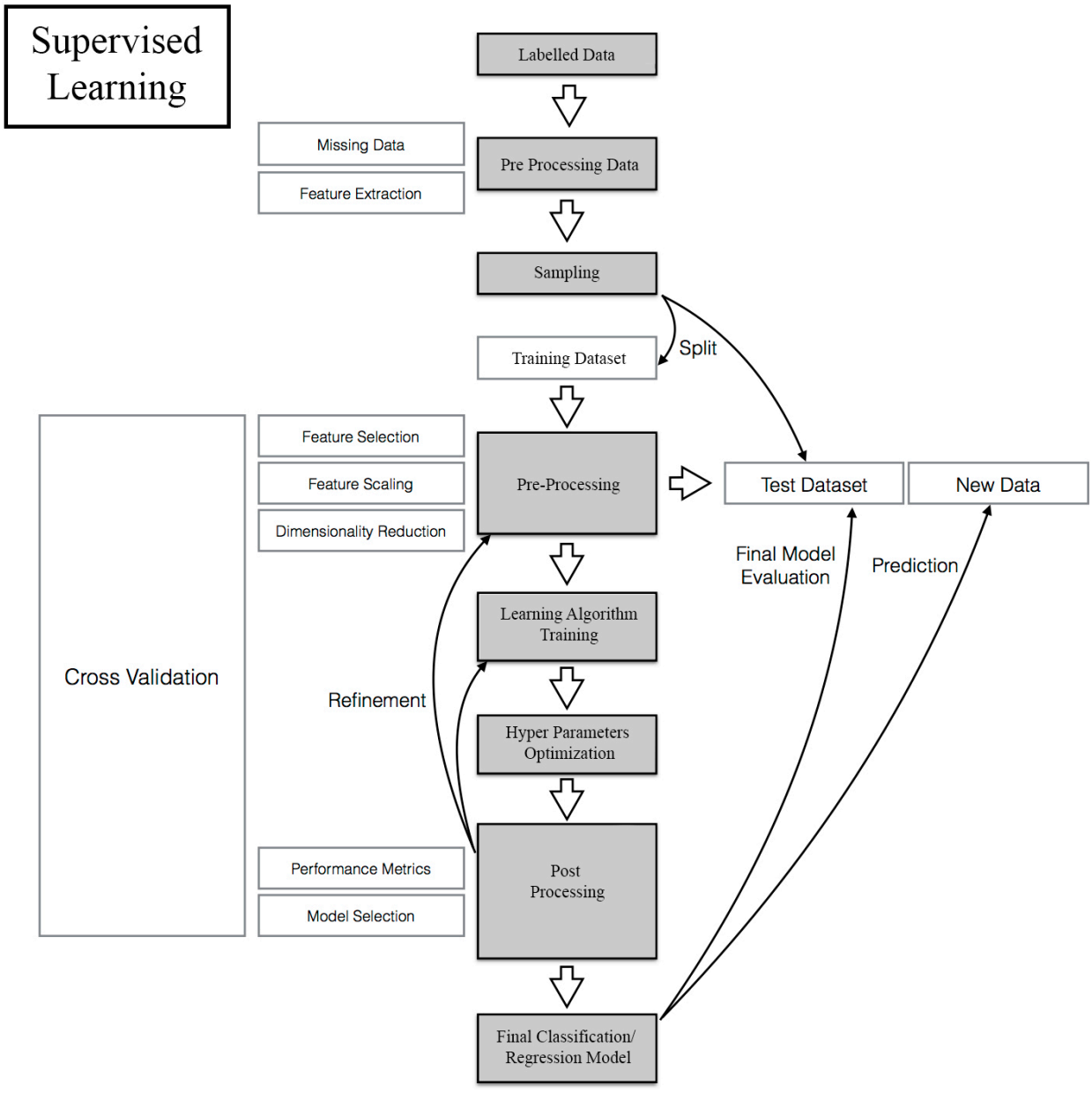


Figure 3.6 Supervised Learning Algorithm

3.4.1 Labelled Data

In supervised learning, to train the model, first the dataset is labelled which is an important feature. As we took the food dataset from Kaggle that's why the dataset was already labelled. In case of Location Prediction Model, we directly took the model from MIT places so no labeling was needed in this case. We choose RGB colored image having input size of 299x299x3 where 299x299 is the width and height of the image while 3 represents (Red – Blue – Green) colors. The Inception networks also expect images scaled to be between 0 and 1, which means that the pixels values needed to be divided by 255 (the maximum intensity value for a colour). The colour image is input to the convolutional neural network for automatic feature extraction and training. Colour image can increase the accuracy of the system because a grey scale image is usually 8bit image with each pixel having 256 combinations of shades of grey. Whereas colour image usually is a 24-bit image with 8bits of Red, 8 bits of Green, 8bits for blue information. Combination of these three basic colours can create 16,777,216 colour combinations for a pixel. That's why if we convolve a greyscale image over RGB, the model accuracy will be much lower.

3.4.2 Pre-processing Data

We first put all our images together, and then randomize the ordering. We did not want the order of images to affect what we learnt, since this is not the part of determining whether the image taken is of food or not. If it is a food image, then we need to determine the name of that food. In other words, we decide of what type of food it is, independent of the order of images such as what type of food became before or after it. In this step, missing data is found and also the features of the dataset are being extracted. Missing data means that when a data set arrives the data has some missing values, either because it exists and was not collected or it never existed. Most of the testing and training images includes noise, intense color and wrong labels. We labeled the testing and training images respectively. Also, images from all the classes have been rescaled to a unique size of 299x299 dimensions. In case of Location Prediction Model, we directly took the model from MIT places so no preprocessing of data was needed.

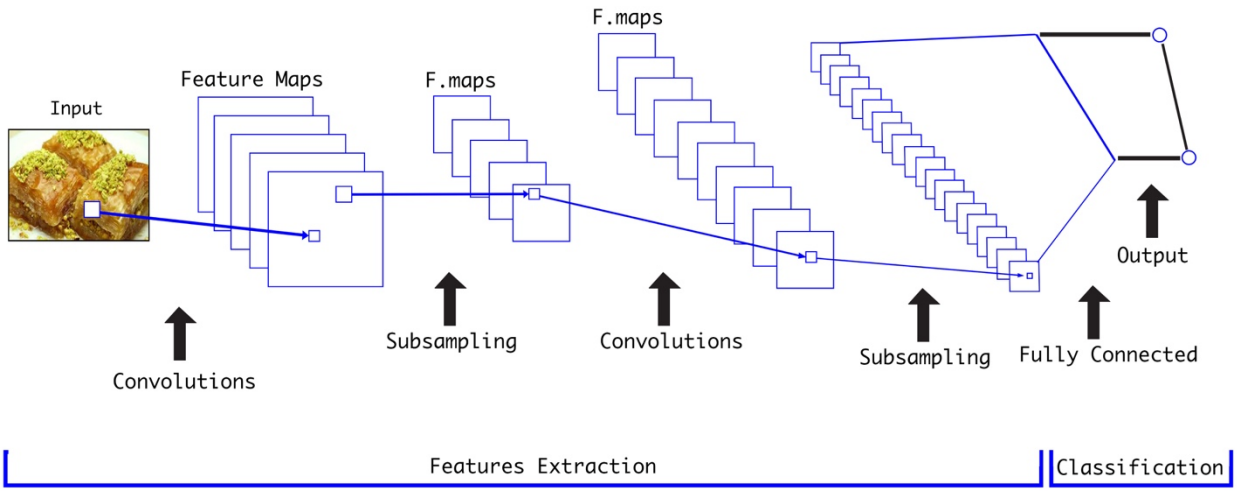


Figure 3.7 CNN for food classification model

As we can see from Figure 3.6, the convolutional network is usually divided into two parts. One for extracting the features and other for the classification. Through convolution layers and subsampling layers' features are extracted.

Particularly the layers used in our convolutional neural network are:

a) **AvgPool Layer**: An AveragePooling2D function (group size (8,8)) is used. The main function of AvgPool layer is to reduce the variance and computational complexity (Dehmer, 2013) of the data. This layer extracts the features without problems.

b) **Convolution Layer**: In this specific layer feature maps are created mainly by convolving the input data. We used a Convolution2D function and set its input size to (299,299,3).

c) **MaxPool Layer:** The main function of MaxPool layer is to reduce the variance and computational effort. We used MaxPooling2D function and its grouping mainly extracts the most important features such as the edges of the data.

d) **Concat Layer:** This layer mainly concatenates the multiples input blobs in to one single output blob. As an input, list of tensors is used, all having the same expected shape for the concatenation axis, and returns a single tensor for the concatenation of all the inputs.

e) **Dropout Layer:** Dropout is a method designed to reduce excessive adaptation in neural networks and it prevents complex adjustments of training data. This is a very effective method to perform model averaging with neural networks. We define dropout scale to be 0.4.

f) **Fully connected Layer:** This layer connects each neuron from one layer to each neuron in another layer.

g) **Softmax Layer:** The use of the Softmax function (Zheng, 2018) as an output function works almost as a maximum level and it is also possible to practice through gradient descent. In addition, the sum of all the outputs will always be 1.0.

3.4.3 Sampling

In this step the dataset is divided into two sets.

i. Training Dataset
ii. Testing Dataset

3.4.4 Image Pre-processing Parameters

In order to ensure the maximum efficiency of the system, image preprocessing techniques are used which can classify any type of image. Here are the parameters that are considered for pre-processing images.

a) Rotation range = 45: The images are rotated at random 45 degrees. This make sures that the images taken with each angle can be correctly predicted and preserve the variety of feature maps obtained.

b) Width shift range = 0.2: The images are shifted horizontally by this fraction. This makes it possible to predict "incomplete" or "half" images, and the patterns obtained will differ.

c) Height shift range = 0.2: It is having the same purpose as horizontal shift.

d) **Horizontal flip = True:** The images are reflected horizontally. By rotating the images at random, different patterns can be detected and images can be predicted accurately.

e) **Fill mode = reflect**: Out of range points are filled in this mode.

3.4.5 Learning Algorithm Training

All images are resized to a size of 299 x 299 x 3. The dimensionality of the output space is defined by the dense function. The dropout rate of 0.4 on input units is considered to avoid overfitting problems. To determine the actual class of n classes, the Softmax activation function is defined. Identify the class according to the maximum probability that will be obtained in the output of this class and ignore the rest. The model is trained for 32 epochs and has three callbacks that record the progress in a log file. Learning rate is defined. It uses the epoch index as input and returns a new learning rate at the output. These are saved as. hdf5 files. The size of food model is 87mb while the size of location model is 298mb. Our IOS mobile application size is 300mb.

3.4.6 Final Classification

So finally, after the classification of the dataset we attain a trained model which is now ready to be tested with new images. Particularly, users can upload images using the IOS application, then the system predicts the food or spots the location on google maps.

3.4.6.1 Food prediction:

image that the user uploaded was related to any food item, the user will be shown the image with the name of the food and the accuracy in percentage.

3.4.6.2 Location Prediction:

In case if the image that was uploaded was related to any monument or any place the application will automatically predict the best three spots on google maps.

CHAPTER 4
SOFTWARE DESIGN OF PROPOSED SYSTEM

Nowadays, the development of mobile applications is attracting increasing interest. The rapid increase of mobile and smart devices in the consumer market has forced the software engineering community to quickly adapt conscious development approaches to the new capabilities of mobile applications. The combination of computing power, access to new built-in sensors, and ease of application transfer to the market has made mobile devices the new computing platform for businesses and independent developers. Here we will be discussing the software design methods (Diaz, 2005) for the proposed Machine Learning Based IOS Mobile Application.

Specifically, we examine the challenge of:
a) Creating user interface accessibility.
b) Handling the complexity of providing application on IOS.
c) Designing of the application,
d) Specifying software requirements.

4.1 Data Flow Design of the proposed Application

According to different studies, mobile application users express themselves in a way that they are doing something in their daily life routine. The personality of a mobile application user can be extracted from the fact that how is the user using the application. A person who is an extrovert in real life always tends to search a lot and will be using the application more than a neurotic who often tends to be less active and uses a smaller number of searches. The data flow of the application in this study was built on the architecture of the application. Here we explain the complete flow of the IOS application in Figure 4.1. The flow chart explains how the user will open the application and how the user will receive results from the food or location predictor. Our mobile application only comes with the convolution model and not with images. As the size of the dataset is 6.5 GB while the size of application is merely 300 MB.

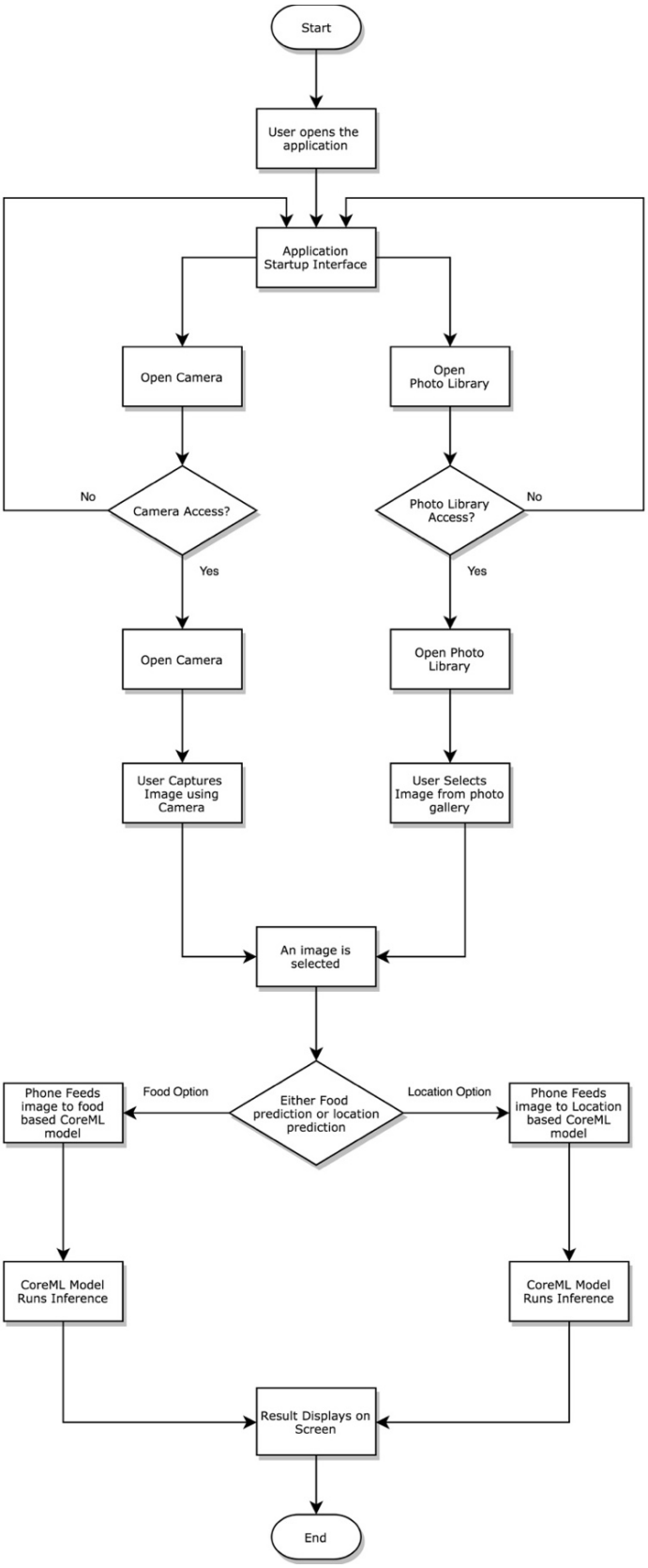


Figure 4.1 Data flow design of our application

As we can see when the user opens the application the interface will be displayed. The user will be prompt to upload the image. There are two options to upload the image, either by uploading the image or taking a picture through the camera. When the user opens the camera or image gallery for the first time, the user will be asked for permission. After the user allows the usage of camera or photo library, he/she can upload or take the image. After this step, user will be asked if the image uploaded is related to food or location that he or she just uploaded.

4.2 Entity Relationship (ER) Diagram of Proposed Application

An entity relationship diagram (ERD) shows the relationships of entity sets stored. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties. User will have "Access" to all photos which will be "used for" prediction. Application can be used without internet connecting as well if trying to predict food Images. If predicting location Images, internet is required to access google maps. Here is the complete ER Diagram of the mobile application.

4.2.1 User

The entity user has different attributes which are required for a user: Apple Mobile Phone, Installed Application and Internet for Google Maps.

4.2.2 Photos

The Photos entity has different attributes which are required for accessing the photos: Camera and Photo Gallery.

4.2.3 Prediction

The Prediction entity has different attributes. The prediction can be performed for two different entities: Food and Location

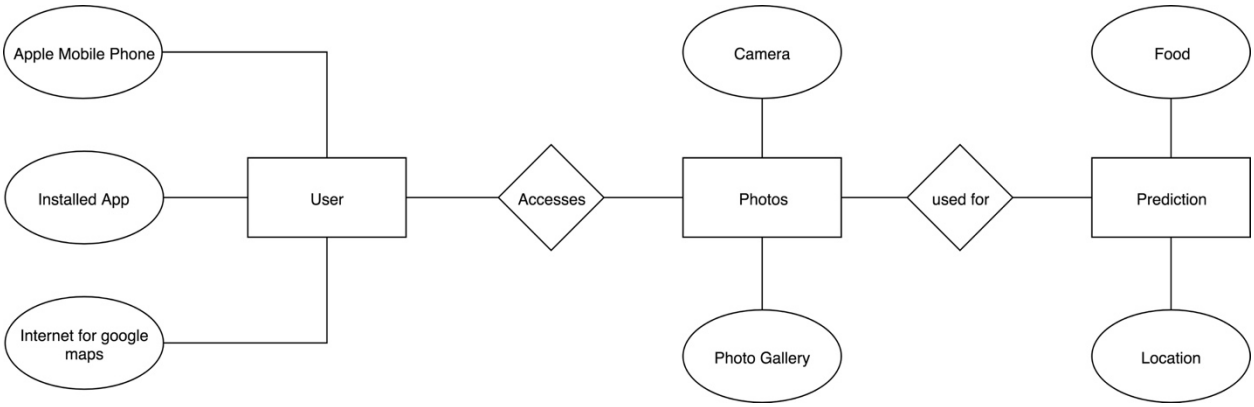


Figure 4.2 ER Diagram

4.3 Conversion to CoreML

The food dataset we used for the application is taken from Kaggle and then the prediction is done using the API library of Tensorflow, that is Keras and a machine learning model is generated. When the model is developed then the next step that come is converting the model to coreml using XCode IDE. coreml is the machine learning framework for the XCode and used to integrate the models into Apple Application. The model is integrated in to XCode and then the layout is designed in XCode and after coding in swift the application is ready to use.

As a first step, we need to convert our existing food and location-based model to CoreML's which has. mlmodel extension. For this we need to install **coremltools**. In order to install coremltools we run the following command on the command line.

```
pip install -U coremltools
```

CoreML tools require Python 2.7 and above. For this process we setup a virtual environment with Python 2.7. We used Anaconda Navigator. After Initializing Anaconda Navigator, we clicked on "Environments", as shown in figure 4.3. Finally, we click on the create button (Figure 4.4) under the list of environments.

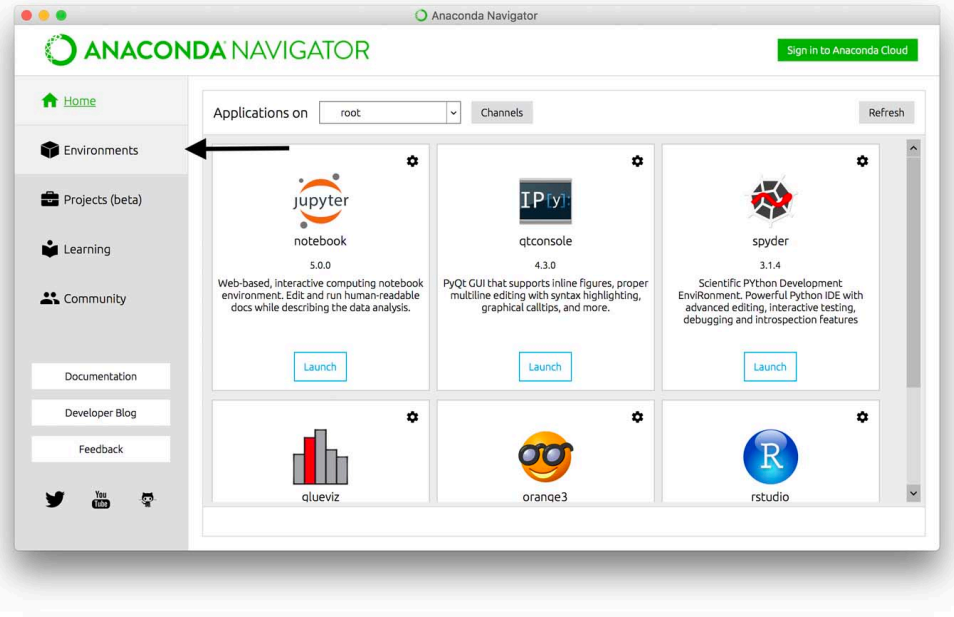


Figure 4.3 Anaconda Navigator

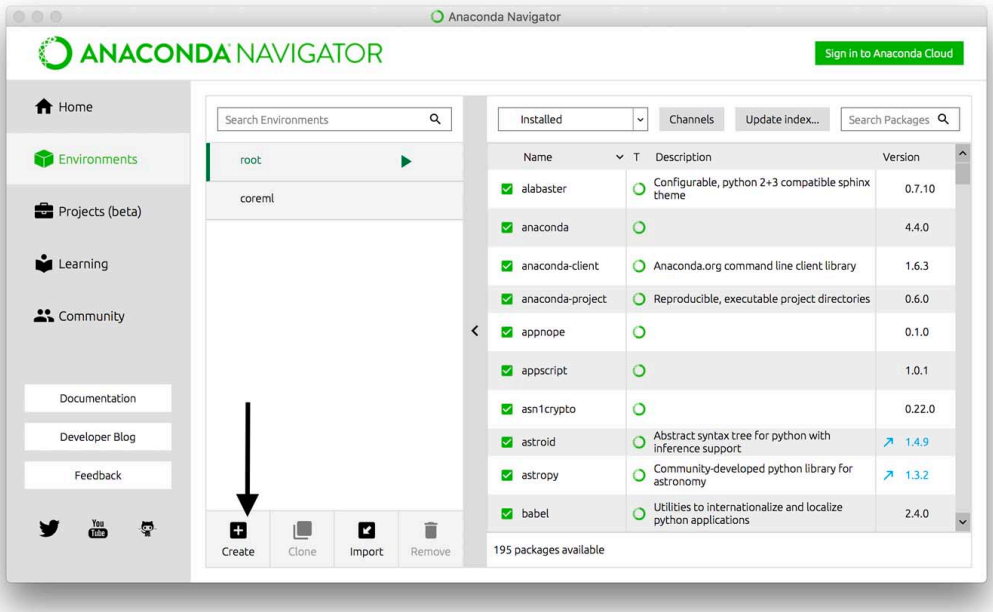


Figure 4.4 Environments in Anaconda Navigator

In Order to create a new Environment, we type “coreml” for name, and checked Python and selected “2.7” from the dropdown (Figure 4.5). Following this step, the new “coreml” environment is up and running after few seconds. After that if click on the new environment and press play, a new command line is opened (Figure 4.6).

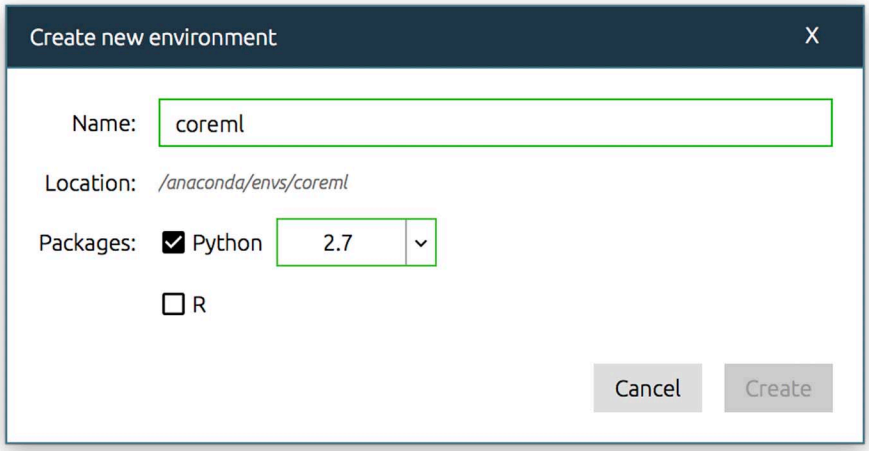


Figure 4.5 Creating a new environment in Anaconda Navigator

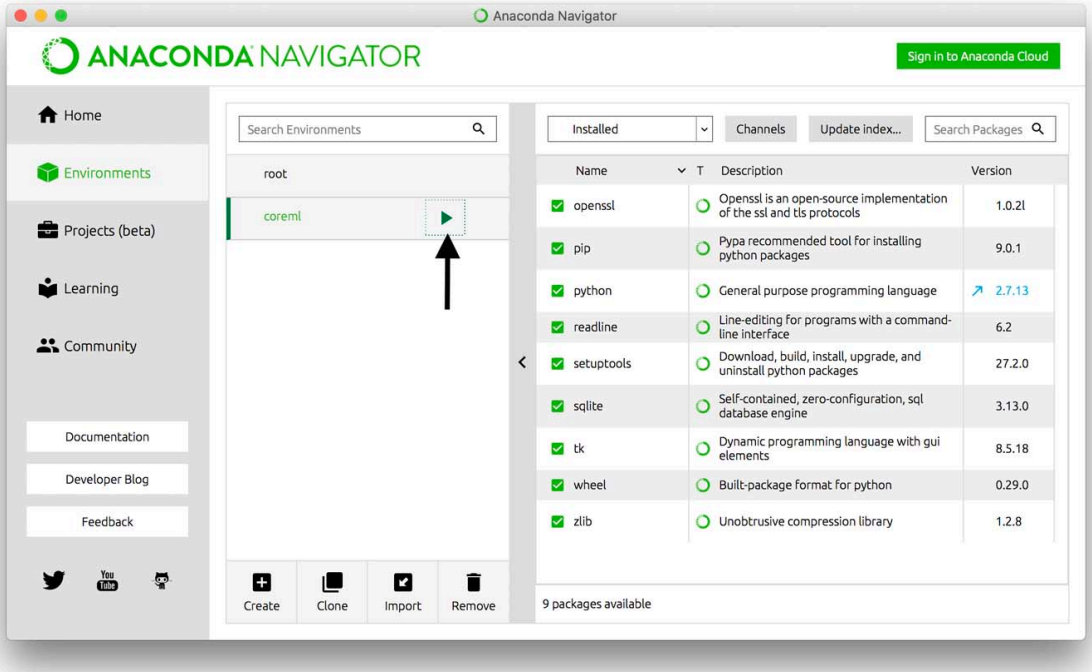


Figure 4.6 Finishing step for creating a new environment in Anaconda Navigator

Subsequently the following commands can be run as follows:

After creation of new environment, we ran the following commands. **pip install -U coremltools** from our new command line. A new file, that is named **run.py** is created which contains the following code.

```
import coremltools
# Convert a caffe model to a classifier in Core ML
coreml_model = coremltools.converters.caffe.convert(('snapshot_iter_24240.caffemodel',
'deploy.prototxt',
'mean.binaryproto'),
image_input_names = 'data',
 class_labels = 'labels.txt',
 is_bgr=True, image_scale=255.)
# Now save the model
coreml_model.author = "Owais Qayyum"
coreml_model.save('food.mlmodel')
```

Figure 4.7 CoreML Model Conversion

Coremltools are imported in the first line**.** After we imported, a CoreML model is created and provided with all necessary inputs for coremltools to convert the model to coreml using the Caffe framework (Wei, 2017). Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is used for converting the machine learning model to coreml model. After this step, we integrate trained CoreML model into an IOS project as explained in the next section.

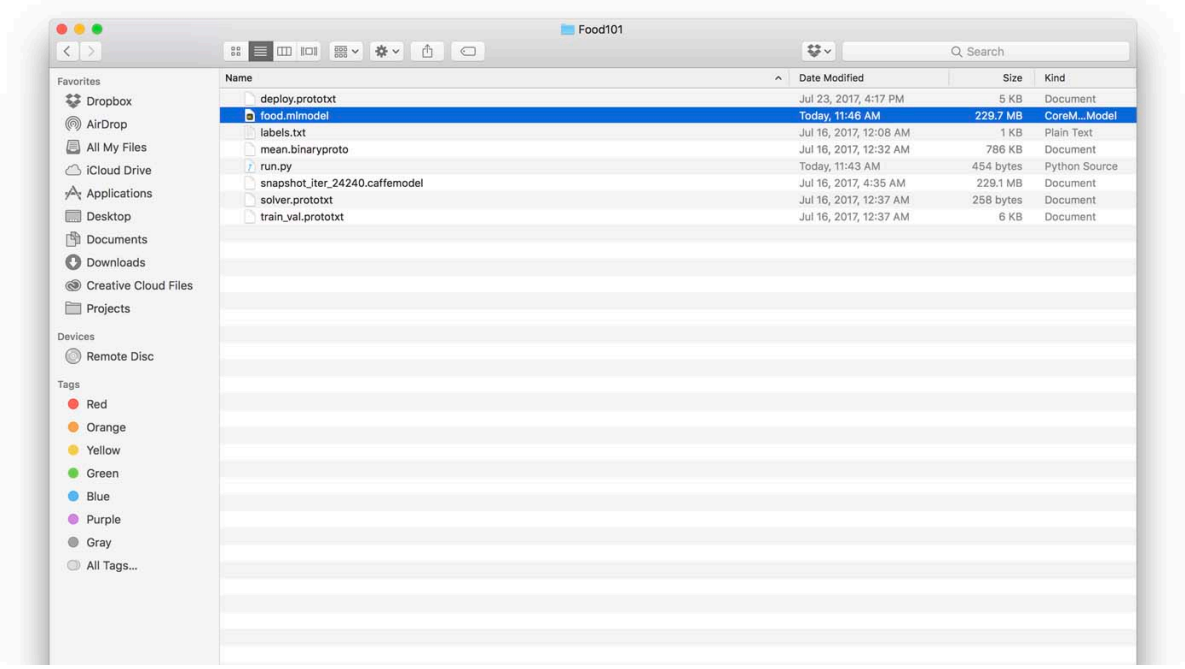


Figure 4.8 CoreML Model

4.4 Integration of CoreML Model into IOS

We started a new Xcode project and selected IOS as a Single View App. We named the proposed application as “pixify” and clicked next. Then we selected a desired folder where we wished to create the project and clicked Create (Figure 4.9).

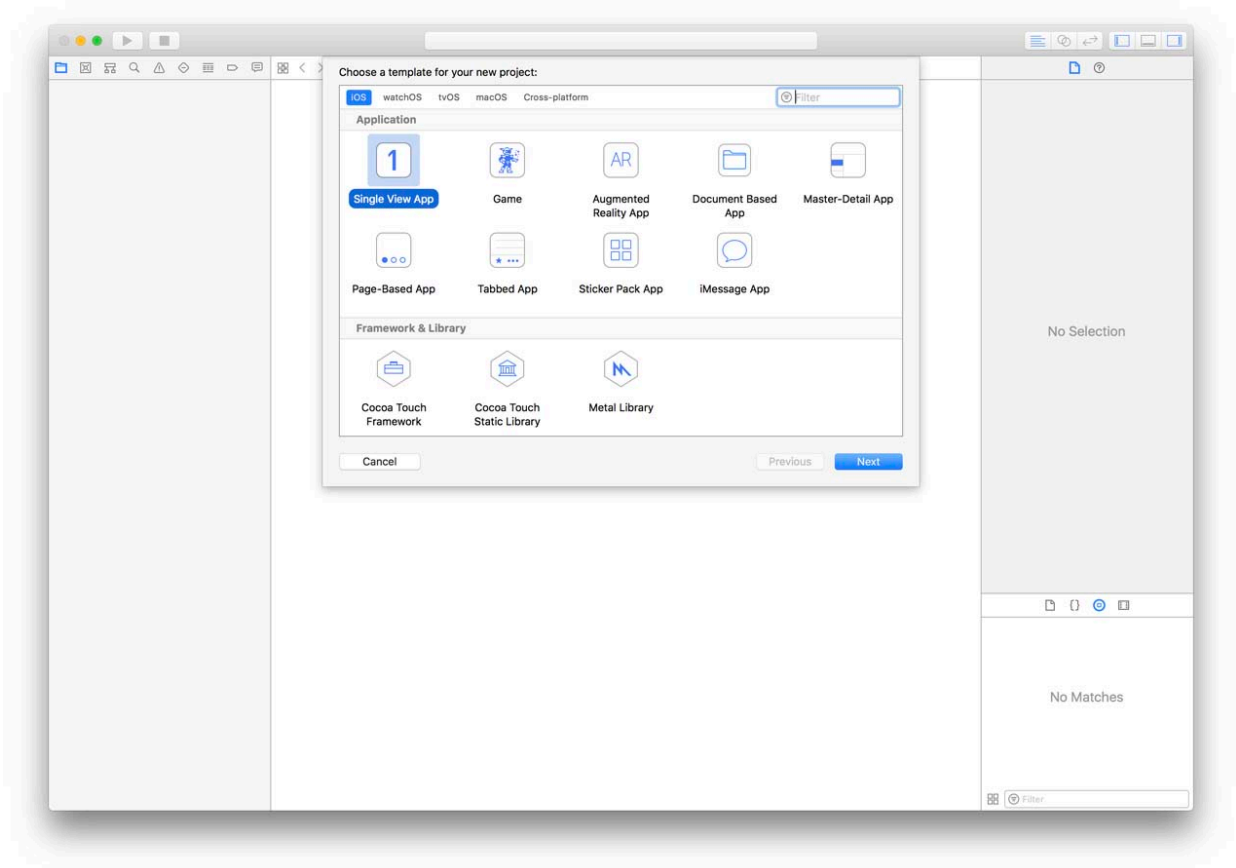


Figure 4.9 Starting a new IOS project in Swift

Once we created the project by right clicking on the "Pixify" folder under Project Navigator on the left, we can add files to "Pixify". When the model is imported into Xcode, after a few seconds **food.mlmodel** should appear under the Project Navigator. The model details, are shown in figure 4.11 as follows.

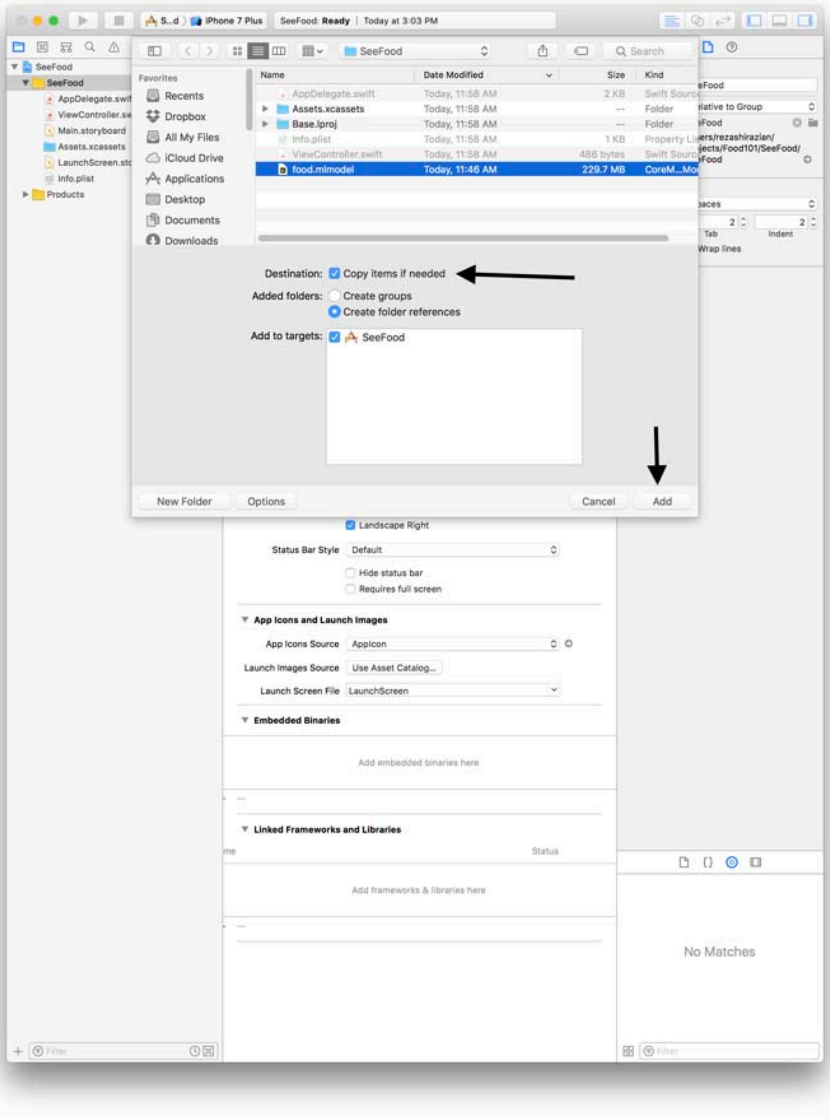


Figure 4.10 Importing the model into Xcode

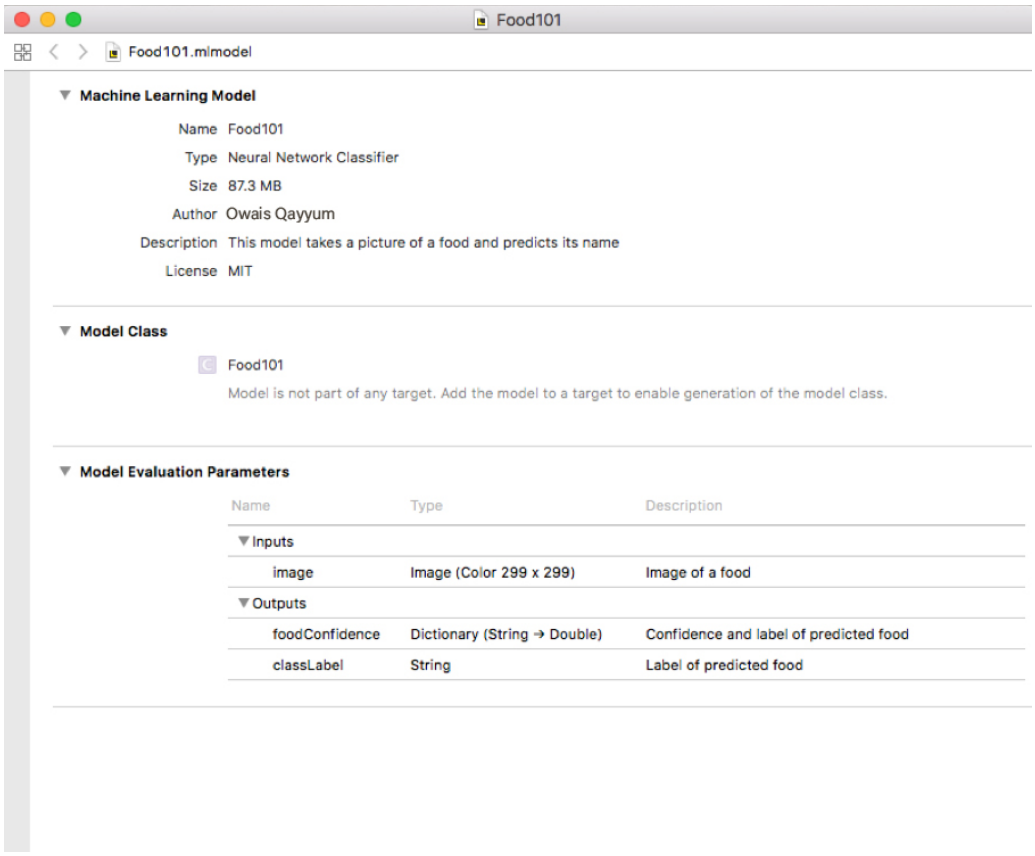


Figure 4.11 Food Model

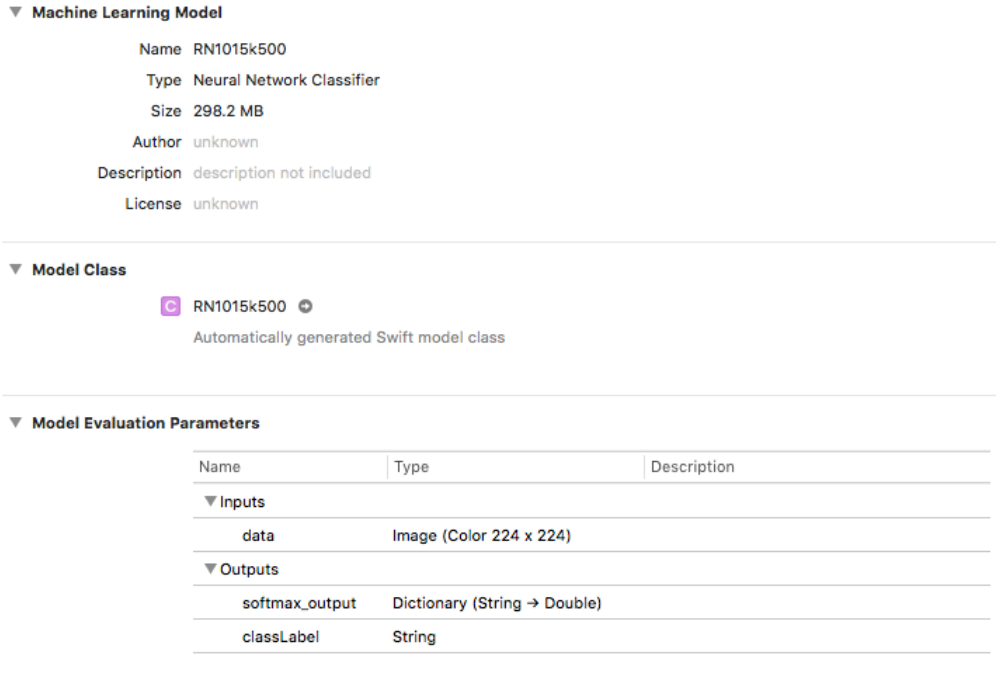


Figure 4.12 Location Model

4.5 User Interface of the IOS App, Pixify

The layout of the application which was named as Pixify is shown below:

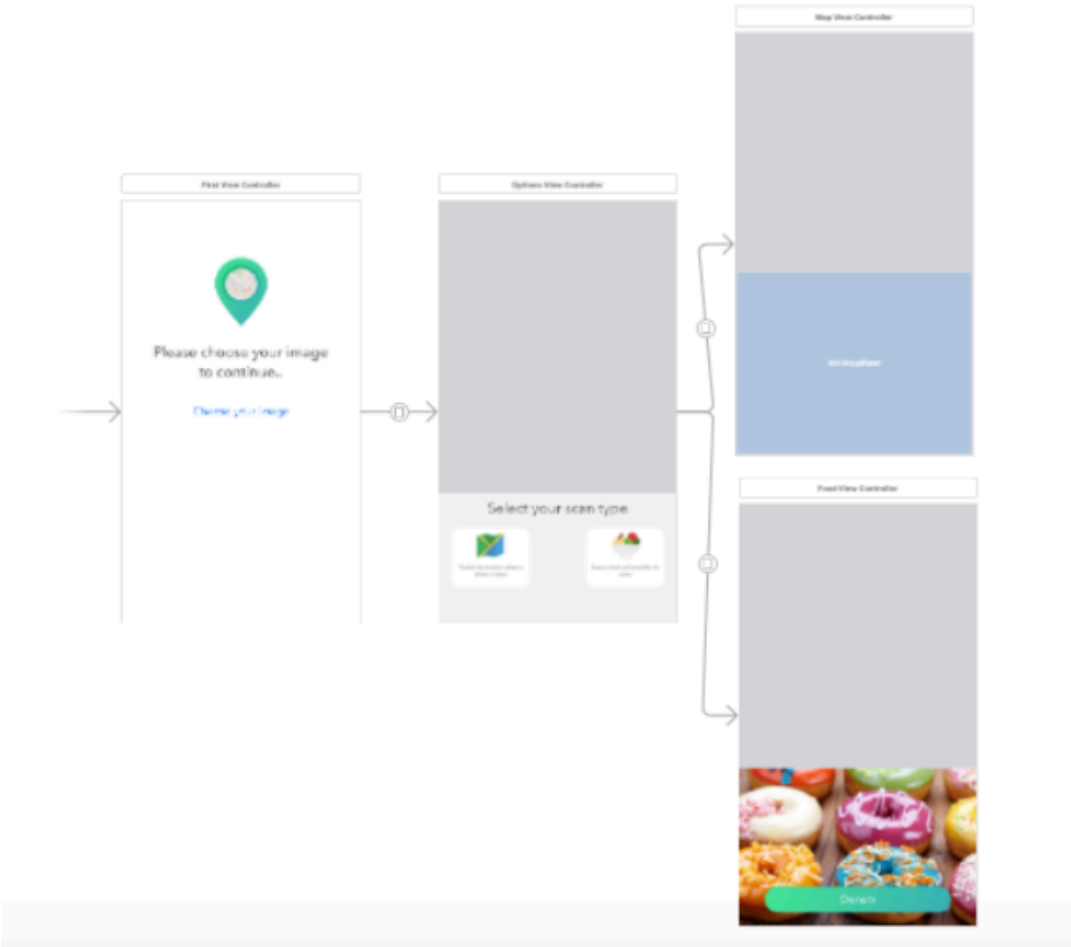


Figure 4.13 Main Story Board

First the application is opened and the layout of the application is shown (Figure 4.15). After that you have two options either to upload the image directly or to open camera for taking the image of your desired food, as shown In Figure 4.14.

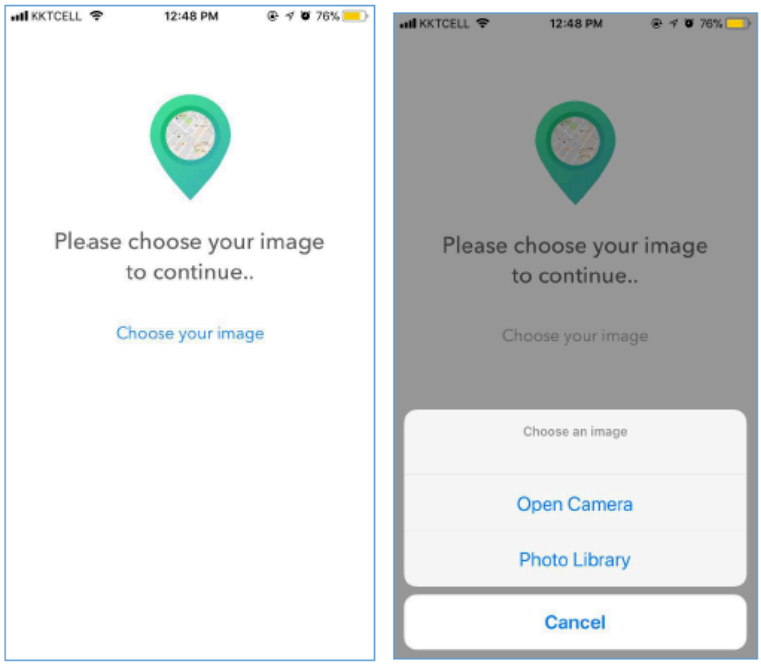


Figure 4.14 Pixify Application User Interface

In Figure 4.16 upon uploading or taking image of your favorite food, you will be shown two options. For scanning the food, you will click on "scan the food button".

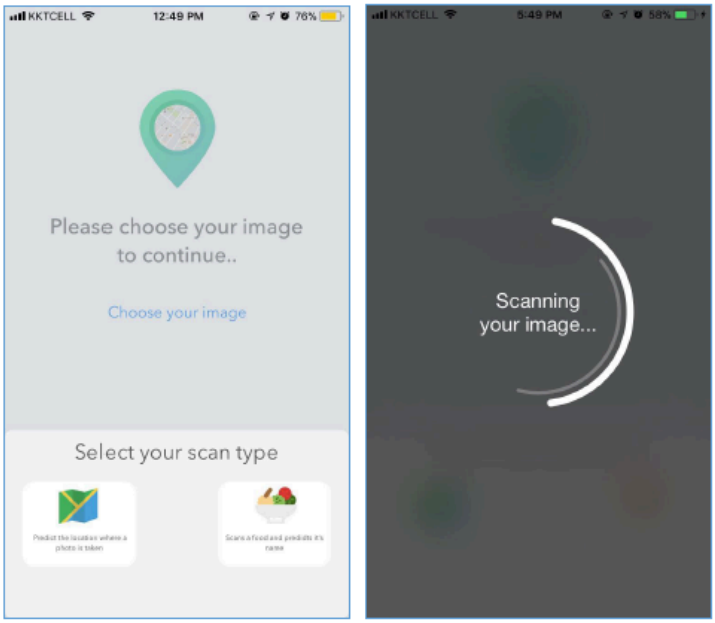


Figure 4.15 Scanning food or Place

Finally, the interface displays you the name of the food with percentage, which are calculated automatically by the model. In Figures 4.17 to 4.20 sample outputs of the Pixify interface are shown for food prediction and location prediction. In the location predictor, we have integrated google maps API (Shaw, 2017) in swift code which takes the longitude and latitude values for the predicting three best spots and displays on the IOS Application. Here is the snippet of google API used in IOS application.

```
var google_maps_api: String = "AIzaSyDFDralfncnU_8xZOkU-V4V43ASjpkGc "

   func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {

      google_maps_api = "AIzaSyDFDralfncnU_8xZOkU-V4V43ASjpkGc"

      save_the_key(key: google_maps_api)

      return true
```

Figure 4.16 Google Maps snippet in Swift

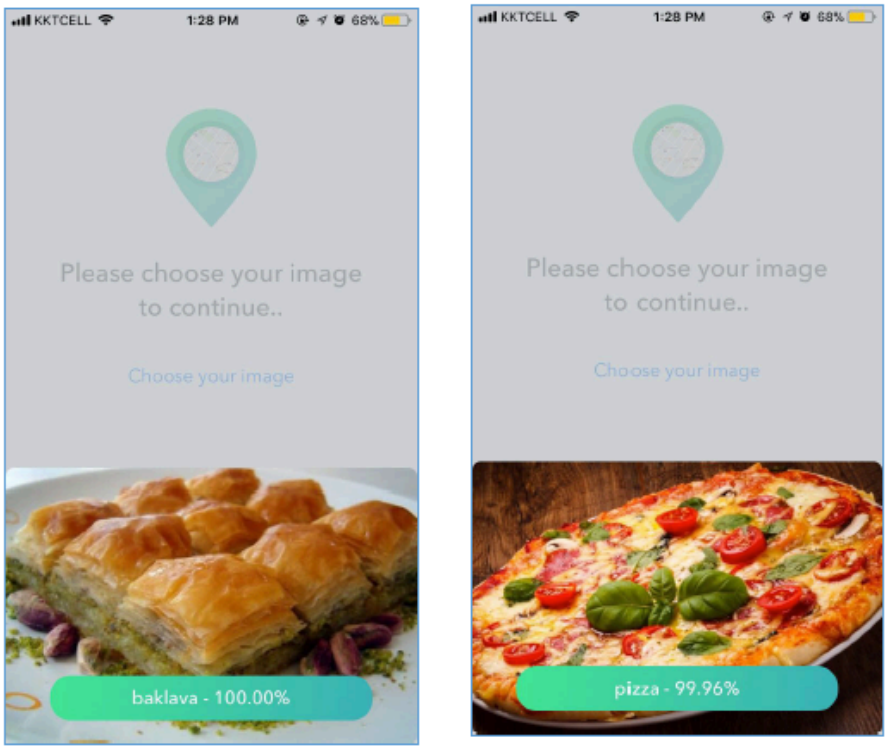


Figure 4.17 Pixify application scanning a Turkish delight baklawa and a pizza.

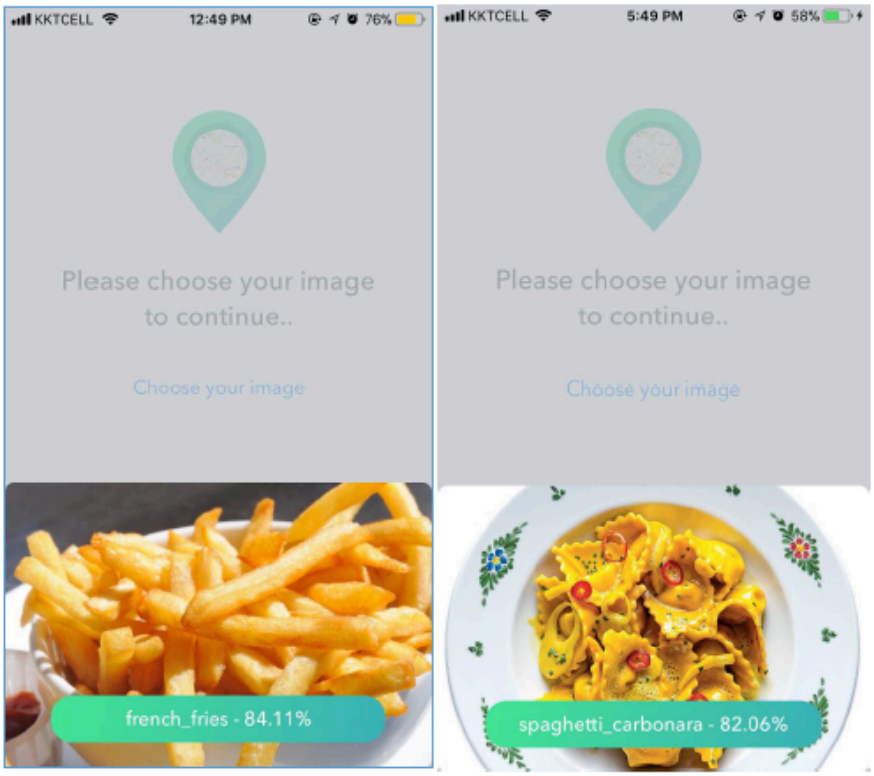


Figure 4.18 Pixify application scanning French fries and spaghetti carbonara

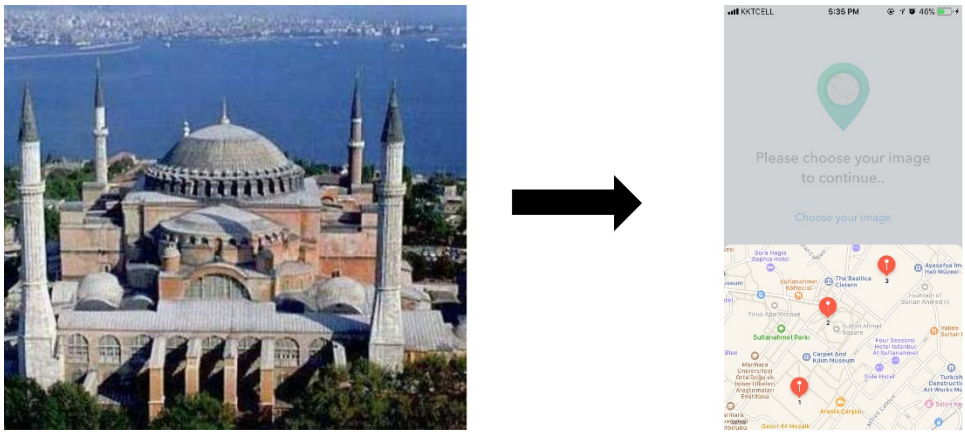


Figure 4.19 Pixify application predicting three best possible locations for Hagia Sophia on google maps

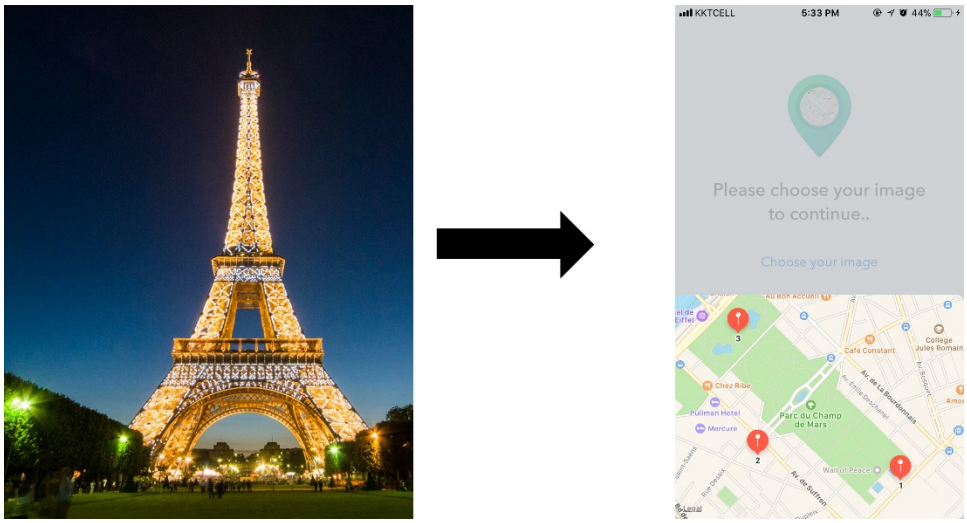


Figure 4.20 Pixify application predicting three best possible locations for Hagia Sophia on google maps

CHAPTER 5
CHOOSING CNN MODEL FOR IOS APPLICATION

With the help of machine learning, we determined different type of food images and show them in percentage and vice versa. Here we applied some principles of machine learning in order to get the required output. These steps for achieving the required output are given below:

- Data Collection
- Data Preparation
- Model Selection
- Dataset Training
- Evaluation
- Hyperparameter tuning
- Conversion to CoreML
- User Interface
- Results or Outputs

5.1 Food or Location Prediction Algorithm

In this thesis we created a prediction based IOS application in which the mobile application will be detecting the food names with the accuracy of predicting food images in terms of percentages. This prediction will be done for all the food classes. The same applies to the location images, user takes an image of a place and upload it to the IOS application. Application automatically compares it with the dataset of 5 million images and shows three closest locations on google maps. The fig 5.1 shows the basic architecture of our food and location based mobile application. The tools we have used in our project are:

a. Python Language
b. Anaconda for creating, training and testing the model.
c. Tensor flow a machine learning library.
d. Core ML for converting the pre-defined model to CoreML model so that it can be used in swift for application development.

e. Convolution Neural Network which the branch of deep neural network (Ramampiaro, 2018) for training and testing the dataset.

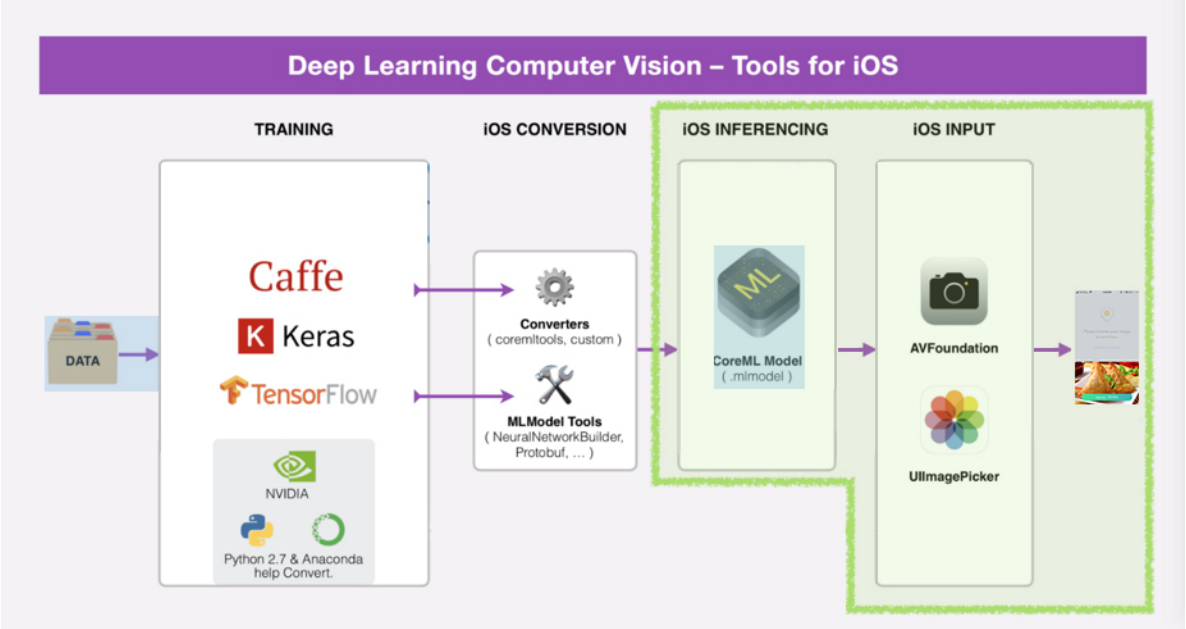


Figure 5.1 Deep Learning Computer Vision for our research. From Data selection to viewing it on IOS application

In the above figure as we can see that there are various steps from taking the dataset to viewing the results on IOS application.

5.2 Data Collection:

First, we took the data set of about 5000 food images from FOOD – 101 and 4 million location-based images from MIT Places. First of all, we gather all the data and then ordered them randomly. We did not want the order of our data to be affected what we learned, since that was not the part of determining whether the image taken is of food or not. If its food, then what is the name of that food. Same in the case of location prediction, we were trying to evaluate that where the picture that is been uploaded or taken is located on the google maps. In other words, we decide of what type of food it is, independent of what type of food image came before or after it. Same with the location prediction, it is dependent on either that place is available in the dataset or not.

We split the data into two parts, one is for training the machine learning model and other for testing the model. The model will be trained in such a way that the data points collected from all the food types must be same otherwise the train model will be biased towards a specific food type and not for the others. We will not the same training data for testing the model as the model is built on the training dataset. Hence, we will be evaluating the model with the testing dataset so that the evaluations should be performed on unseen image samples

5.3 Choosing a Model

In this step we will be choosing the model. With time, researchers and data scientist have created and tested many models. Among them some are well suited for image processing, others for sequences, some for numerical data, and others for text-based data. In our case, since we are dealing with images only so we choose an image-based model and in this case, it is InceptionV3 model**.** InceptionV3 is one of the models to classify images. We have used TensorFlow and Keras. InceptionV3 model is a model of keras framework which is used for prediction, feature extraction, and fine-tuning. The default input size for this CNN model is 299x299. In case of location prediction, we choose the pre-trained model from MIT places which we converted into coreml using coreml tools i.e. Anaconda IDE and Keras framework.

5.4 Training the model

In this step, we used data to enhance our model's ability to predict the name and percentage of a particular food image. In addition, the top three locations in Google Maps for an image based on the location. While training, some parameters are used to ensure the maximum efficiency of the system.

5.5 Testing the Model

Once training is completed, we then used the assessment to determine the quality of model. This is where the testing dataset is used. Evaluation allows us to test our model with some arbitrary data that has never been used while training the model. With the use of different parameters, we can analyze how the model works with unseen data. We then divided training score of 75/25 or 80/20. This means that 75% of the data is learning data or training data and 25% of the data is test data

and vice versa for the 80/20 case. As far as location model is concerned there was no need of testing the model.

5.6 Parameter Tuning

Hyper parameters were tuned to get the best results. These hyper parameters are Rotation range (The images are rotated at random 45 degrees), Width shift range (The images are shifted horizontally by the fraction of 0.2), Height shift range (It is having the same purpose as width shift), Horizontal flip (By rotating the images at random, different patterns can be detected and images can be predicted accurately), Fill mode (Out of range points are filled in this mode), Random crop size (Assign the cropping size of images sent to the network, in this case 299x299x3).

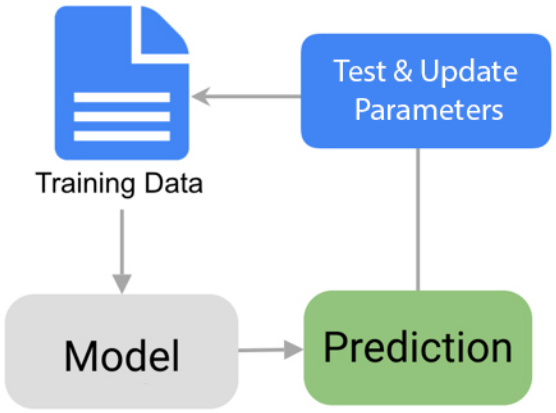


Figure 5.2 Iteration for hyper parameters on Training step

These parameters are usually called "hyperparameters". The adaptation or adaptation of these hyperparameters is still somewhat artistic and constitutes an experimental process that strongly depends on the specificities of the data set, the model and the training process. Once satisfied with the hyperparameters and training, the model is finally ready to do something useful.

5.7 Prediction

Machine learning uses data to answer different questions. Certain questions need to be answered at the prediction stage.

5.7.1 Food Prediction

As we can see from Figure 5.1 modeling technique discussed in Figure 5.2 is applied on the test data which then perform the prediction and the result is shown.

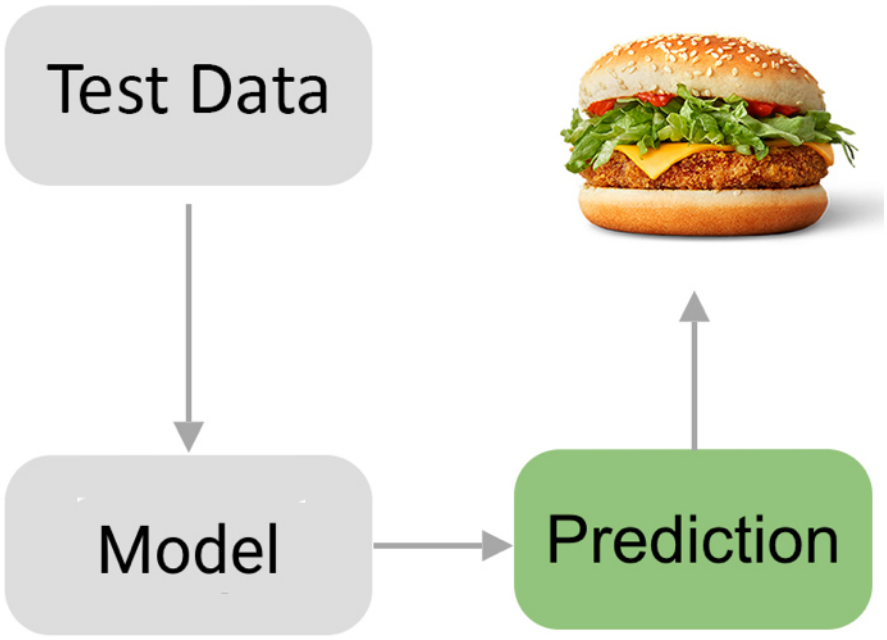


Figure 5.3 Food Prediction Model

Finally our model can be used for prediction, whether a data element of a particular food item is a hamburger, beef or any other food.

5.7.2 Location Prediction

The same applies to the location predictor, where the image of a location has been scanned by the application. A convolutional neural network mechanism displays the first three points on Google maps.

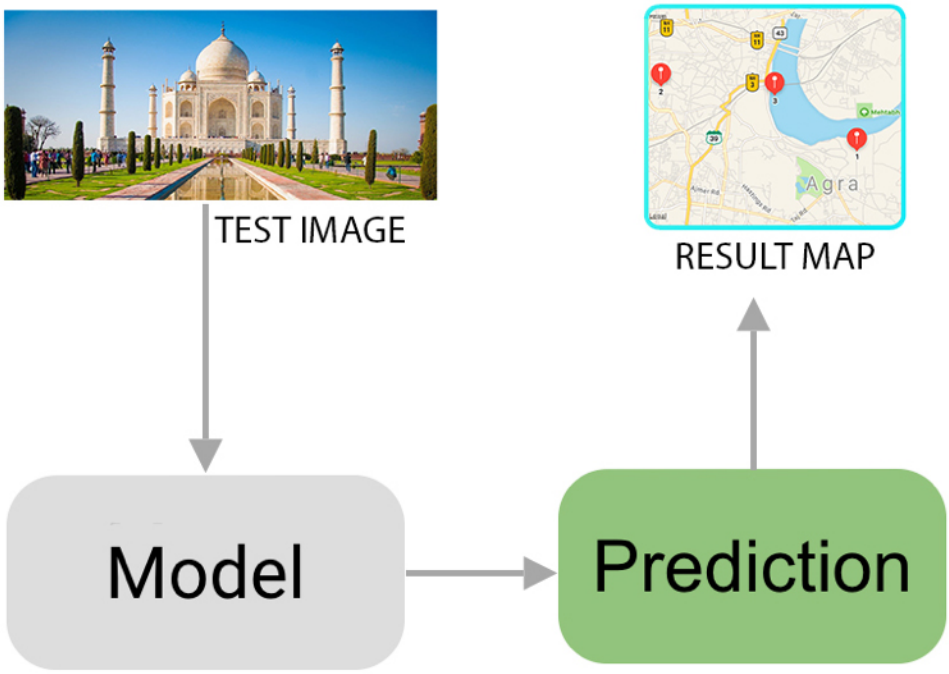


Figure 5.4 Location prediction model

CHAPTER 6
PERFORMANCE EVALUATIONS

Here in this chapter we will be discussing various aspects of the mobile application performance. These aspects include the time taken for classification of an image into its respective class and time taken for an image to be loaded and scanned etc. In addition, we will evaluate classification accuracy of the deep learning model.

6.1 System Performance

For the system evaluations, we took 101 food classes each having 750 training images and 250 testing images. The proportion of test train split for our food dataset was 3:1 per class. For the location data set we took a prebuilt model from MIT places which originally contained 5 million location images. The test train split used for the location dataset was 4:1 which means that 4 million were training images and 1 million testing images. As we are dealing with images only so we choose an image-based model and in this case, it is InceptionV3 model**.** InceptionV3 is one of the models to classify images. We have used TensorFlow (Hope, 2017) and Keras. InceptionV3 model is a model of keras framework which is used for prediction, feature extraction, and fine-tuning. Size of food model is 87mb while the size of location model is 300mb.

Table 6.1 Food and location datasets

Dataset	Training Images	Testing Images	Proportion	Size of Model
Food Dataset	750 images per class	250 images per class	3:1	87 mb
Location Dataset (CoreML)	4 Million	1 Million	4:1	300 mb

User will input an image to the coreML model using his camera or gallery and after the inference from coreML model, results will be displayed on the screen. As we can see from figure 4.1. the complete process from loading up the image and displaying the results of the proposed IOS application.

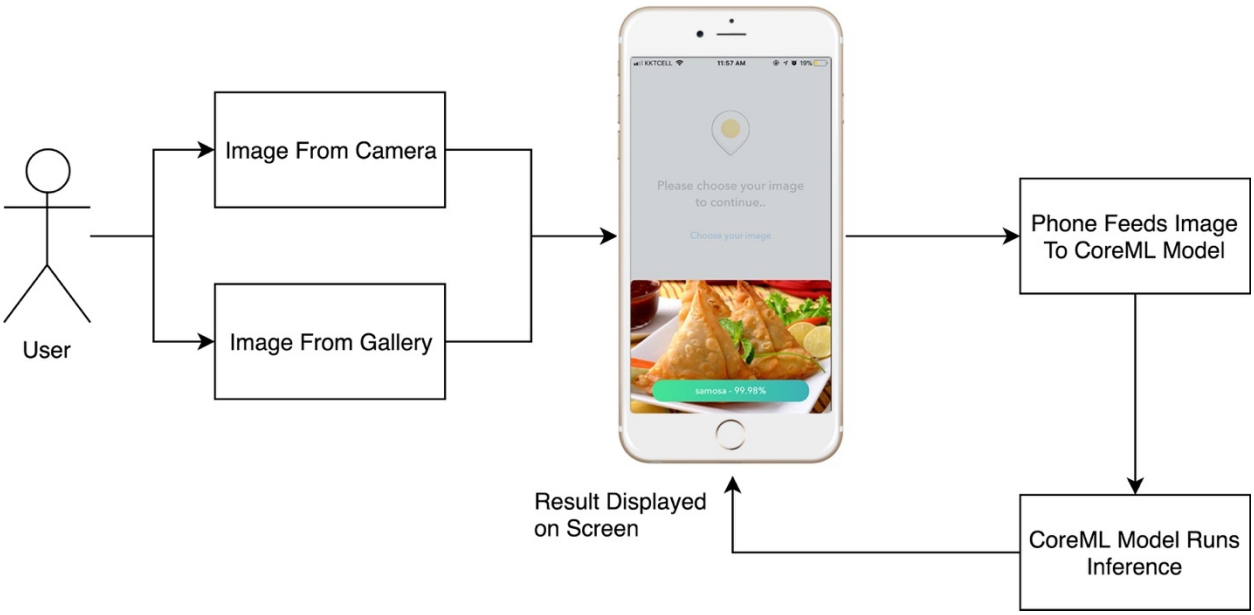


Figure 6.1 Time for Loading/Scanning Images

The time taken for training the model depends on different things. These includes size of the dataset, speed of the device, ram of the device and the processor of the device. The specifications of system that we used for training the dataset on inception V3 model is shown in table 6.2.

Table 6.2 Specifications of System used for Training the Dataset

Device	Specifications
Macbook Pro	2017
Ram	8 GB
Processor	2.3 GHz dual-core Intel Core i5
eDRAM	64MB
Graphics	Intel Iris Plus Graphics 640
Frequency	50Hz – 60Hz

6.1.1 Time for Loading/Scanning Food Images

When a user is using a mobile application, time is the main power and for that reason we analyze the time taken by different class images to be scanned and loaded with the respective percentage

of accuracy. On the food class images, we repeated three iterations and receive an average time in seconds for each class to load the food image. Here in Figure 4.2 we can see how the distribution is taken place. On average it takes 3 seconds for loading or scanning food images and for some images it takes up to 5 seconds to scan and load. Some food classes might take more time than the other because of the less features to be extracted while other classes have more features to be extracted. For this analysis we used iPhone 6 having 1 GB of RAM and dual core 1.4GHz processor.

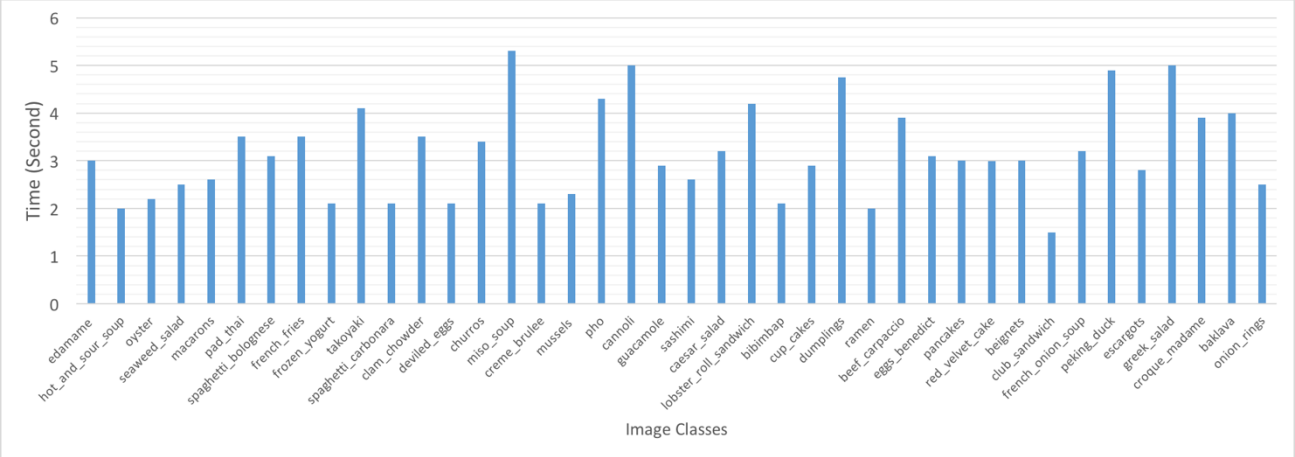


Figure 6.2 Time for Loading/Scanning Food Images

6.1.2 Time for Loading/Scanning location Images

Similarly, for the location-based images, we conducted the same test so that we can investigate the time required to predict three best spots on google maps. For this evaluation, we tested 50 location images from different parts of the world and ran our results on it. We analyzed different images of the same monument three times. Results are shown in figure 6.3. Generally, it takes 5 to 7 seconds for loading or scanning an image. As location images take more time for loading or scanning because of the fact that location model requires internet.

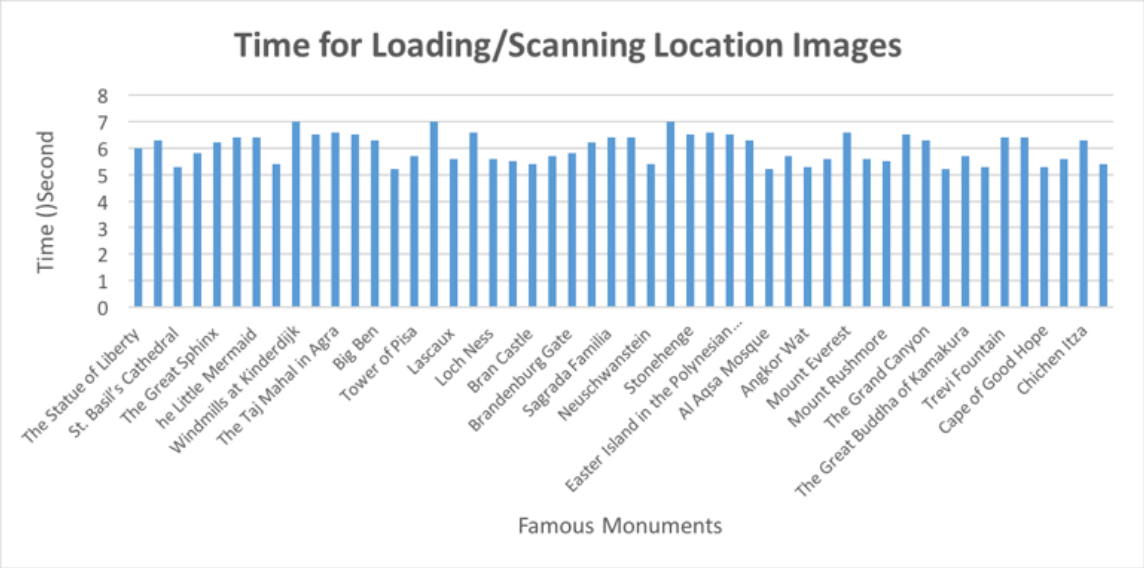


Figure 6.3 Time for Loading/Scanning Location Images

6.1.3 Time for classifying food images on Average using CNN

Usually the time taken for classifying the images using CNN depends on different features. These features include various things such as the size of the dataset, speed of the device, ram of the device and the processor of the device. The processor and ram specifications of iPhone 6, 6s, 7, 7s, 7plus, 8, 8s and X are shown in table 6.3.

6.1.4 Iphone Specifications

Iphone specifications are shown in Table 6.3.

Table 6.3 Specifications of IOS devices used for Analysis

Device	Ram	Processor
Iphone 6	1 GB	Dual core 1.4 GHz
Iphone 6s	2GB	A9 dual-core
Iphone 7	2GB	A10 Fusion
Iphone 7 plus	3GB	Apple A10 Fusion
Iphone 8	2GB	A11 Bionic
Iphone 8 plus	3GB	A11 Bionic
Iphone X	3GB	A11 Bionic

The time require in seconds for classifying the images on average using convolutional neural network may differ from one to device to another. Here we have tested the model on different devices and here are the results displayed.

Table 6.4 Average Time for classifying the image using CNN

Class Name	**Iphone 6**	**Iphone 6s**	**Iphone 7**	**Iphone 7 Plus**	**Iphone 8**	**Iphone 8 Plus**	**Iphone X**
edamame	6.3	5.5	4.8	4.2	3.8	3.5	3.1
hot_and_sour_soup	6.6	5.6	4.9	4.3	4.0	3.7	3.3
oyster	6.4	5.4	4.7	4.2	4.1	3.8	3.4
seaweed_salad	6.2	5.3	4.8	4.4	4.0	3.7	3.2
macarons	6.3	5.4	4.7	4.3	3.9	3.6	3.1
pad_thai	6.9	5.6	4.6	4.5	4.0	3.7	3.5
spaghetti_bolognese	7.1	5.8	4.5	4.4	4.1	3.6	3.2
french_fries	5.3	5.3	4.8	4.3	4.2	3.8	3.3
frozen_yogurt	7.3	5.4	4.8	4.2	4.1	3.7	3.5
takoyaki	5.3	5.4	4.9	4.2	4.0	3.6	3.2
spaghetti_carbonara	7.5	5.5	5.0	4.3	4.2	3.7	3.1
clam_chowder	5.7	5.6	5.1	4.5	3.9	3.7	3.4
deviled_eggs	6.6	5.4	4.8	4.2	3.9	3.8	3.2
churros	6.4	5.3	4.8	4.3	3.8	3.8	3.3
miso_soup	6.4	5.4	4.9	4.2	4.0	3.7	3.1
creme_brulee	6.2	5.6	4.7	4.4	3.8	3.7	3.3
mussels	6.3	5.8	4.8	4.3	3.9	3.5	3.4
pho	6.9	5.3	4.7	4.5	4.1	3.6	3.2
cannoli	7.1	5.4	4.6	4.4	4.1	3.5	3.1
guacamole	5.3	5.4	4.5	4.3	3.9	3.6	3.5
sashimi	7.3	5.4	4.8	4.2	3.9	3.7	3.2
caesar_salad	5.3	5.3	4.8	4.2	3.8	3.7	3.3
lobster_roll_sandwich	7.5	5.4	4.9	4.3	4.0	3.8	3.5

bibimbap	5.7	5.6	5.0	4.5	3.8	3.8	3.2
cup_cakes	6.6	5.8	5.1	4.3	3.9	3.7	3.1
dumplings	6.4	5.3	4.8	4.2	4.1	3.7	3.4
ramen	7.5	5.4	4.5	4.4	4.1	3.5	3.2
beef_carpaccio	5.7	5.4	4.8	4.3	3.8	3.6	3.3
eggs_benedict	6.6	5.5	4.8	4.5	4.0	3.5	3.1
pancakes	6.4	5.6	4.9	4.4	4.1	3.5	3.3
red_velvet_cake	6.6	5.4	4.6	4.3	4.0	3.7	3.4
beignets	6.4	5.3	4.8	4.2	3.8	3.8	3.2
club_sandwich	6.2	5.4	4.9	4.2	4.0	3.7	3.1
french_onion_soup	6.3	5.6	4.7	4.3	3.8	3.6	3.5
peking_duck	6.9	5.8	4.8	4.5	3.9	3.7	3.2
escargots	7.1	5.3	4.7	4.2	4.1	3.6	3.3
greek_salad	5.3	5.3	4.9	4.3	4.1	3.8	3.5
croque_madame	7.3	5.4	4.7	4.2	3.8	3.7	3.2
baklava	5.3	5.6	4.8	4.4	4.0	3.6	3.1
onion_rings	6.6	5.8	4.7	4.3	4.1	3.7	3.4
tacos	6.4	5.3	4.6	4.5	4.0	3.7	3.2
fish_and_chips	6.2	5.4	4.5	4.4	3.9	3.8	3.3
poutine	6.3	5.4	4.8	4.3	3.8	3.8	3.1
cheese_plate	6.9	5.5	4.8	4.3	4.0	3.7	3.3
Sushi	7.1	5.6	4.9	4.5	4.1	3.7	3.4
fried_rice	5.3	5.4	5.0	4.3	4.0	3.7	3.2
chicken_wings	7.3	5.3	5.1	4.2	3.9	3.6	3.1
fried_calamari	5.3	5.4	4.8	4.4	3.9	3.8	3.5
pulled_pork_sandwich	6.6	5.6	4.7	4.3	3.9	3.7	3.2
Waffles	6.4	5.8	4.6	4.5	3.8	3.6	3.3
crab_cakes	6.2	5.3	4.8	4.4	4.0	3.7	3.5
gyoza	6.3	5.4	4.9	4.3	3.8	3.7	3.2
caprese_salad	6.9	5.4	4.7	4.2	3.9	3.8	3.1

paella	7.1	5.4	4.8	4.2	4.1	3.5	3.4
beef_tartare	5.3	5.3	4.7	4.3	4.1	3.7	3.2
samosa	7.3	5.4	4.6	4.5	3.8	3.8	3.3
hot_dog	5.3	5.6	4.5	4.2	4.0	3.7	3.1
shrimp_and_grits	7.3	5.8	4.8	4.3	4.1	3.6	3.3
strawberry_shortcake	5.3	5.3	4.8	4.2	4.0	3.7	3.4
baby_back_ribs	6.2	5.4	4.9	4.4	3.8	3.6	3.2
spring_rolls	6.3	5.3	5.0	4.3	4.0	3.5	3.1
donuts	6.9	5.4	5.1	4.5	3.8	3.7	3.5
lobster_bisque	7.1	5.6	4.8	4.3	3.9	3.8	3.2
prime_rib	5.3	5.8	4.8	4.5	4.1	3.7	3.3
chicken_quesadilla	7.3	5.3	4.9	4.4	4.1	3.6	3.5
hummus	5.3	5.3	4.7	4.3	3.8	3.7	3.2
grilled_salmon	6.6	5.4	4.8	4.2	4.0	3.6	3.1
tiramisu	6.4	5.6	4.7	4.2	4.1	3.8	3.4
pizza	6.2	5.8	4.6	4.3	4.0	3.7	3.2
carrot_cake	6.3	5.3	4.5	4.5	3.9	3.6	3.3
falafel	7.4	5.4	4.8	4.3	3.8	3.7	3.1
ice_cream	5.3	5.4	4.8	4.2	4.0	3.7	3.3
bread_pudding	5.8	5.5	4.9	4.4	4.1	3.8	3.4
huevos_rancheros	7.5	5.6	5.0	4.3	4.0	3.8	3.2
ravioli	5.7	5.4	5.1	4.5	3.9	3.7	3.1
scallops	6.6	5.3	4.8	4.4	4.0	3.7	3.5
chicken_curry	6.4	5.4	4.5	4.3	4.1	3.5	3.2
omelette	6.2	5.6	4.8	4.2	4.2	3.6	3.3
ceviche	6.3	5.8	4.8	4.2	4.1	3.5	3.5
lasagna	7.4	5.3	4.9	4.3	4.0	3.5	3.2
cheesecake	5.3	5.4	4.6	4.5	4.2	3.7	3.1
hamburger	5.8	5.4	4.8	4.2	3.9	3.8	3.4
beet_salad	7.5	5.4	4.9	4.3	3.9	3.7	3.2

risotto	5.7	5.6	4.7	4.2	3.8	3.6	3.3
french_toast	5.3	5.8	4.8	4.4	4.0	3.7	3.1
gnocchi	5.8	5.3	4.7	4.3	3.8	3.6	3.3
garlic_bread	7.5	5.4	4.6	4.5	3.9	3.8	3.4
breakfast_burrito	5.7	5.4	4.5	4.4	4.1	3.7	3.2
chocolate_cake	6.6	5.4	4.8	4.3	4.1	3.6	3.1
steak	6.4	5.6	4.8	4.3	3.9	3.7	3.5
pork_chop	6.2	5.8	4.9	4.5	4.0	3.7	3.2
chocolate_mousse	6.3	5.3	5.0	4.3	4.2	3.8	3.3
apple_pie	7.4	5.4	5.1	4.2	3.9	3.8	3.5
filet_mignon	5.3	5.4	4.8	4.4	3.9	3.7	3.2
foie_gras	6.4	5.5	4.7	4.3	3.8	3.7	3.1
macaroni_and_cheese	6.2	5.6	4.6	4.5	4.0	3.5	3.4
tuna_tartare	6.3	5.4	4.5	4.4	3.8	3.6	3.2
panna_cotta	7.5	5.3	4.8	4.3	3.9	3.5	3.3
bruschetta	5.7	5.4	4.8	4.3	4.1	3.7	3.4
grill_cheese_sandwich	6.6	5.6	4.9	4.5	4.1	3.7	3.2
nachos	6.4	5.8	4.7	4.4	3.9	3.5	3.3
Average Time	**6.5**	**5.7**	**4.9**	**4.5**	**4.0**	**3.6**	**3.3**

As we can see from table 6.3 that when we are testing the application on newer mobile phones such as iPhone X and iPhone 8, the classification time gets faster. From this, we can derive that the time taken for classification of classes and images is directly proportion to the processing power of the device. We downloaded the application of different iPhone devices and tested each class and the derived the results. Average time for classification of food images using CNN can be seen in Figure 6.4.

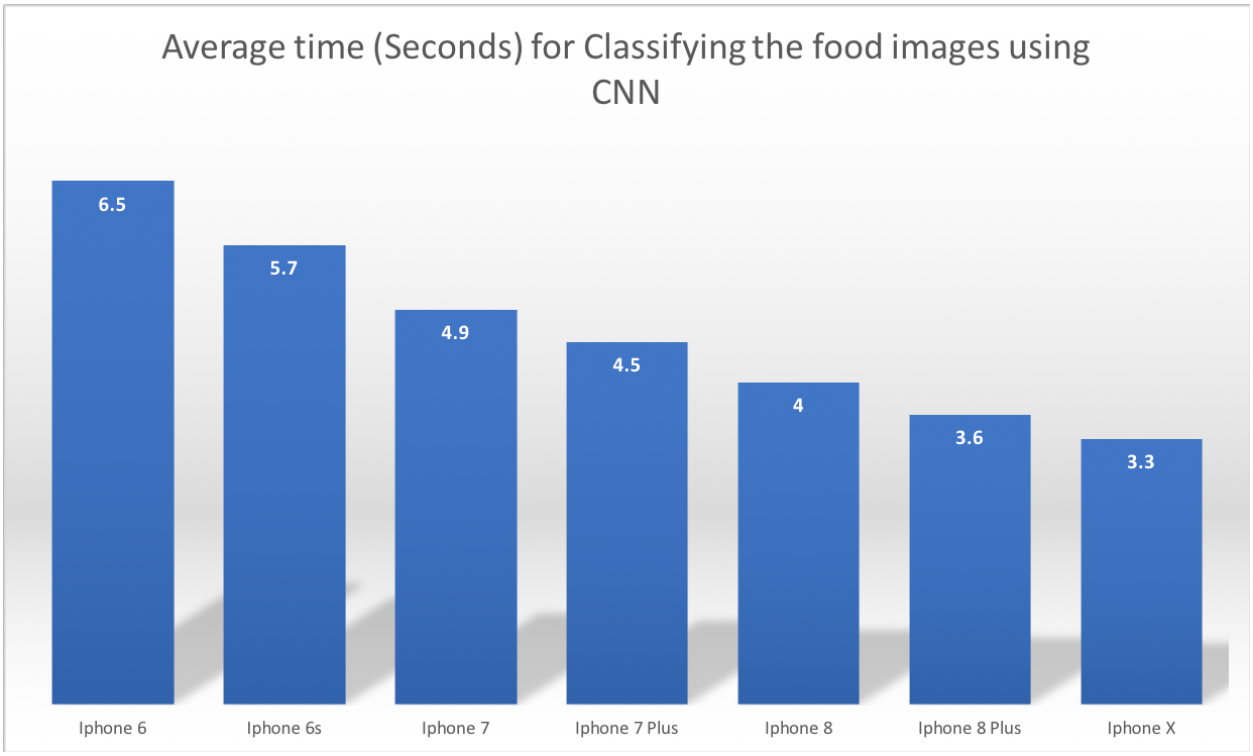


Figure 6.4 Average time in seconds for classifying the food images using CNN

6.1.5 Average Time for Classifying Food Images using CNN

For the food there are 101 classes and upon the search on IOS application it automatically matches the image with the respective class and hence giving us the results using convolutional neural networks. When an image is being scanned through the mobile application it usually takes some time to be classified respectively into the desired class i.e. oyster, pad_thai, apple_pie etc.

In Figure 6.5, the time take for classifying the food images is being shown averagely as each class is tested five times. For some classes, it took more time to classify them.

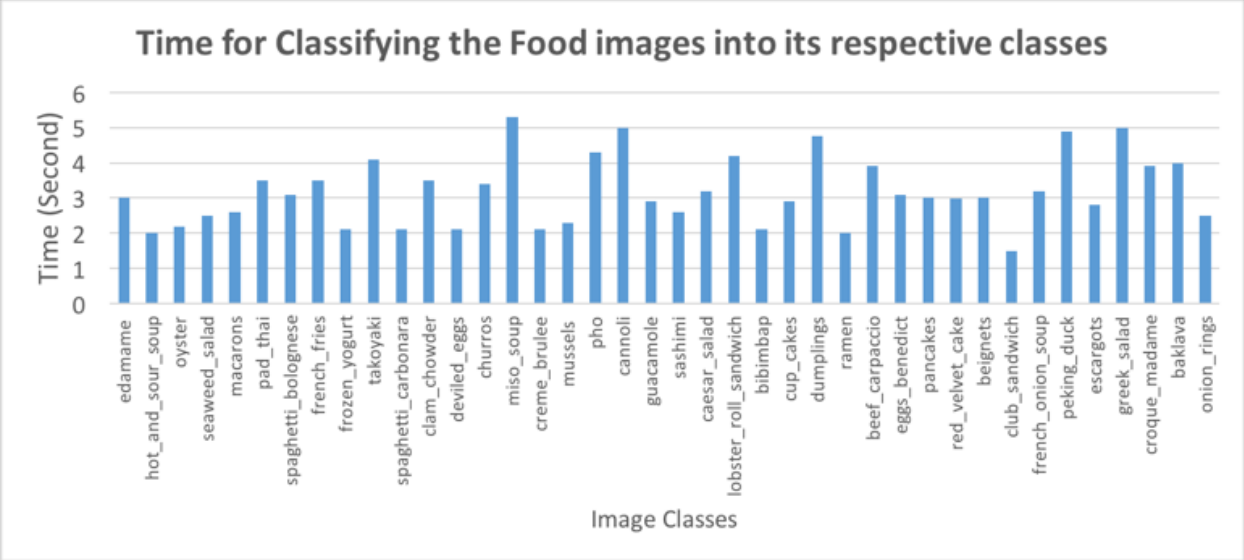


Figure 6.5 Average Time for Classifying the Food Images using CNN

6.1.6 Average Time for Classifying Location Images using CNN

We followed the same procedure for the location images as well and receive some surprising results. The loading time of a map and spotting the best three locations on google map took a bit longer then loading and scanning of food images. In Figure 6.6, we can see the time for classifying the location images on average using CNN (between 5 to 7 seconds).

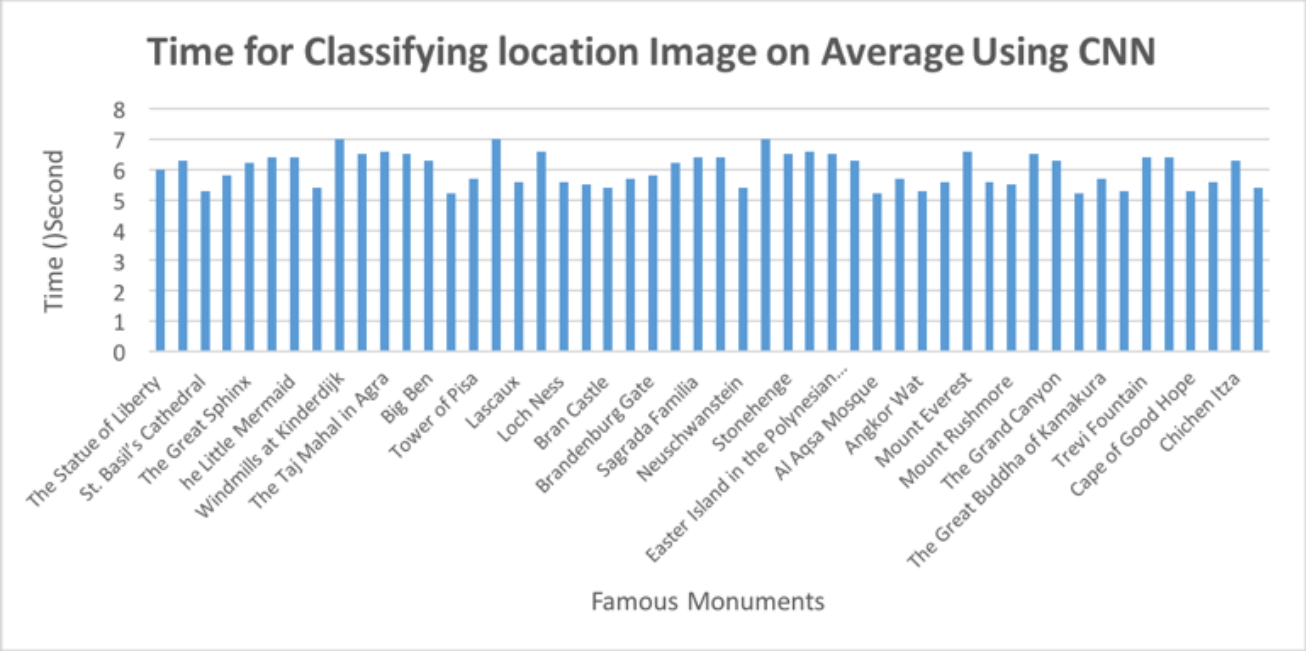


Figure 6.6 Average Time for Classifying the Location Images using CNN

6.1.7 Average Accuracies for predicting the Food Images

We tested the prediction accuracy of some food image classes so that we can get an idea of how well the system is performing. For this evaluation, we tested three different images of the same class and we repeated the phenomena. Figure 6.7 shows the average accuracy of each image class in terms of percentages. Some of the food items such as beef tartare and tacos are showing accuracies around 86 percent and edamame and oyster accuracies were found to be 97 percent. Results are showing the range of accuracy percentages in between 80 and 97 percent. From the Figure 6.7, some classes have low accuracy and some has high. It is because of the fact that some foods are much noisier then the others and not cleaned. For example, if we compare tacos with edemame, we can see that the accuracy of edemame is 97% while that of tacos is 86%.

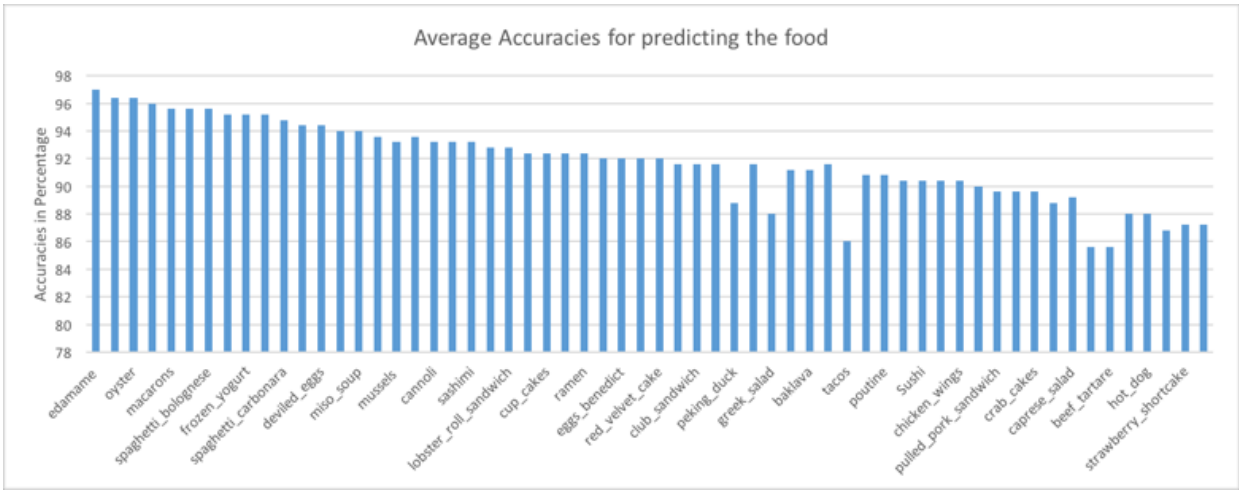


Figure 6.7 Average Accuracies for Predicting the Food Images

6.2 Class Accuracies

Class accuracies of all the 101 food classes are shown in table 6.5 and figure 6.8. Accuracies were increased by cropping each image up to 10 times, as a result accuracy of model was increased by 5%. These results were obtained after training the model. After fine-tuning and training InceptionV3 model, we were able to achieve 82% Top-1 accuracy on the test set using a single crop per item. Using 10 crops per example and taking the most frequent predicted classes, we were able to achieve 86.97% Top-1 Accuracy and 97.42% Top-5 Accuracy. In case of location prediction, no class accuracy was needed as there were no classes involved in location prediction. Location prediction only show three best possible spots on google maps.

Table 6.5 Class Accuracies of all the food classes

Class Name	Accuracies	Class Name	Accuracies
edamame	0.996	Sushi	0.904
hot_and_sour_soup	0.964	fried_rice	0.904
oyster	0.964	chicken_wings	0.904
seaweed_salad	0.96	fried_calamari	0.9
macarons	0.956	pulled_pork_sandwich	0.896
pad_thai	0.956	Waffles	0.896
spaghetti_bolognese	0.956	crab_cakes	0.896

french_fries	0.952	gyoza	0.888
frozen_yogurt	0.952	caprese_salad	0.892
takoyaki	0.952	paella	0.856
spaghetti_carbonara	0.948	beef_tartare	0.856
clam_chowder	0.944	samosa	0.88
deviled_eggs	0.944	hot_dog	0.88
churros	0.94	shrimp_and_grits	0.868
miso_soup	0.94	strawberry_shortcake	0.872
creme_brulee	0.936	baby_back_ribs	0.872
mussels	0.932	spring_rolls	0.876
pho	0.936	donuts	0.876
cannoli	0.932	lobster_bisque	0.888
guacamole	0.932	prime_rib	0.856
sashimi	0.932	chicken_quesadilla	0.852
caesar_salad	0.928	hummus	0.852
lobster_roll_sandwich	0.928	grilled_salmon	0.848
bibimbap	0.924	tiramisu	0.848
cup_cakes	0.924	pizza	0.888
dumplings	0.924	carrot_cake	0.836
ramen	0.924	falafel	0.832
beef_carpaccio	0.92	ice_cream	0.764
eggs_benedict	0.92	bread_pudding	0.748
pancakes	0.92	huevos_rancheros	0.78
red_velvet_cake	0.92	ravioli	0.776
beignets	0.916	scallops	0.784
club_sandwich	0.916	chicken_curry	0.784
french_onion_soup	0.916	omelette	0.784
peking_duck	0.888	ceviche	0.784
escargots	0.916	lasagna	0.792
greek_salad	0.88	cheesecake	0.792

croque_madame	0.912	hamburger	0.796
baklava	0.912	beet_salad	0.796
onion_rings	0.916	risotto	0.812
tacos	0.86	french_toast	0.808
fish_and_chips	0.908	gnocchi	0.808
poutine	0.908	garlic_bread	0.804
cheese_plate	0.904	breakfast_burrito	0.8
filet_mignon	0.716	chocolate_cake	0.78
foie_gras	0.72	steak	0.576
macaroni_and_cheese	0.844	pork_chop	0.676
tuna_tartare	0.832	chocolate_mousse	0.7
panna_cotta	0.828	apple_pie	0.716
bruschetta	0.824		

CHAPTER 7
QUALITATIVE EVALUATIONS

In this chapter we explained two main parts of our IOS application; food images prediction with its respective accuracy in terms of percentage using CNN and location image prediction which will be showing three best spots on the Google maps. For visual inspection, we provided some screenshots of the real time view of our application.

7.1 Food Prediction Qualitative Analysis

For the food prediction system, we took 101 different classes of food images and analyzed those using CNN. Each class is divided into a test train split ratio of three to one. User will be promoted to input the food image of his/her choice either using the camera phone or photo library. When the image is loaded then the user will be asked to choose either to predict the food system or location system. In this case the user will be choosing the food prediction option and below are some of the real time screenshots of the application predicting different food images.

Figure 7.1 Food Prediction Real Time Screenshots of the application

7.2 Location Prediction Qualitative Analysis

Also, for the location prediction system, we took dataset which contains 4 Million images from different countries and then labeled by MIT places according to the nature of the images. This labeled images dataset was then analyzed using CNN. The complete dataset was divided according to test train split with the ratio of 4 to 1. User will be promoted to input the location image of his choice either using the camera phone or photo library. When the image is loaded then the user will be asked to choose either to predict the food system or location system. In this case the user will be choosing the location prediction option and below are some of the real time screenshots of the application predicting different location images with three best choices.

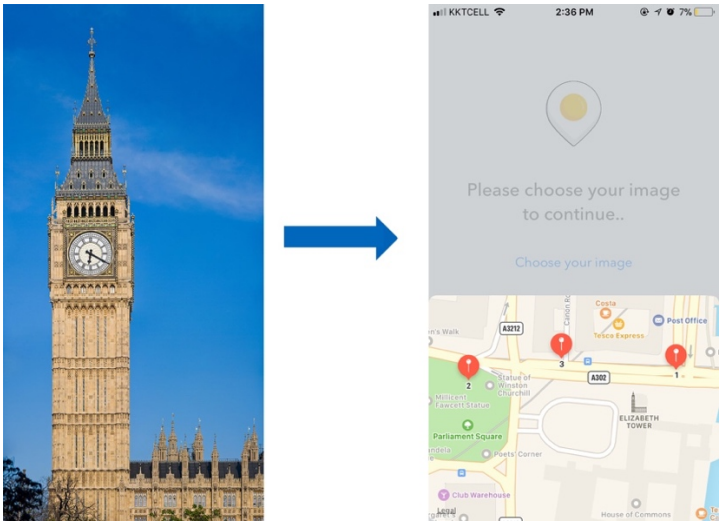


Figure 7.2 Mobile Application predicting Big Ben Tower located in London, UK

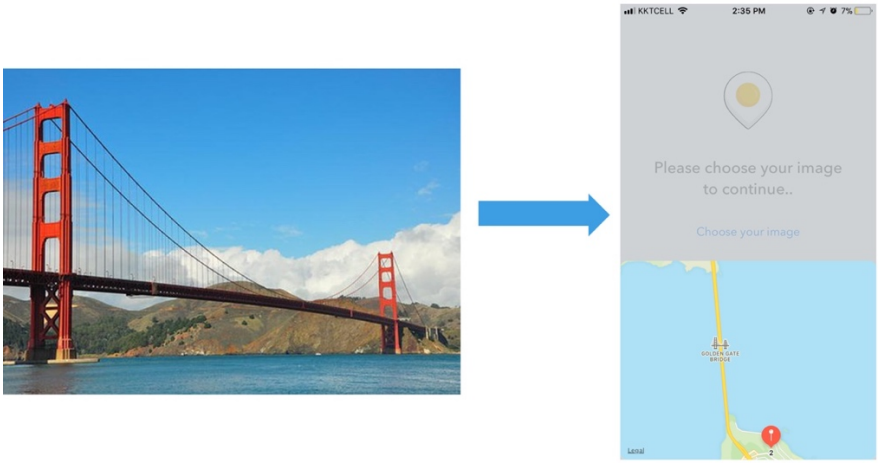


Figure 7.3 Mobile Application predicting Golden Gate Bridge located in San Francisco, USA

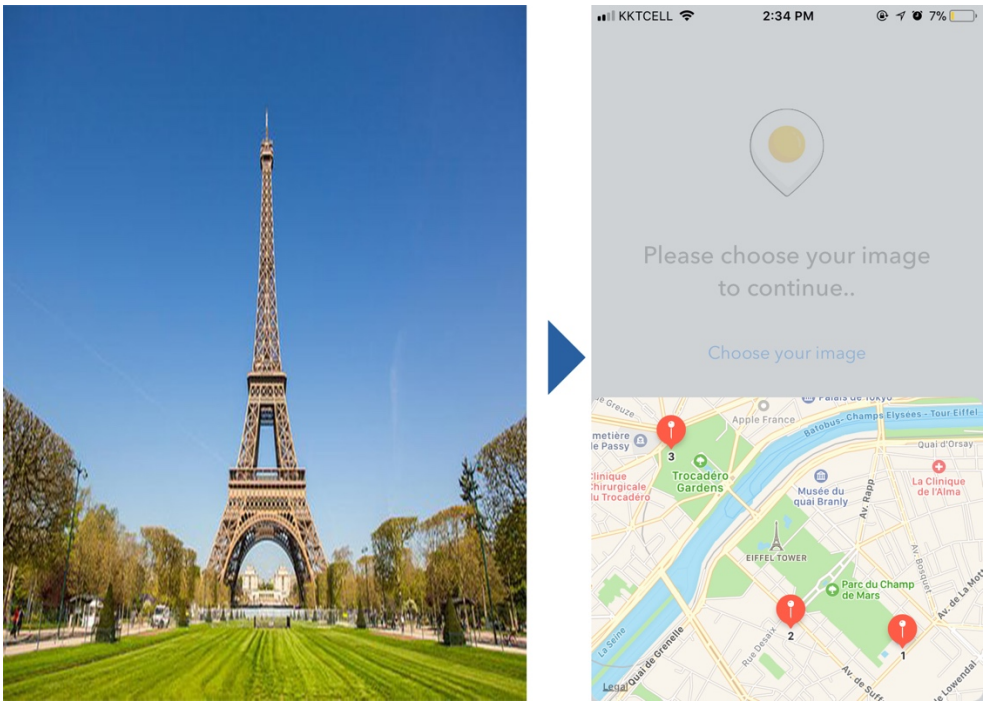


Figure 7.4 Mobile Application predicting Eifel Tower located in Paris, France

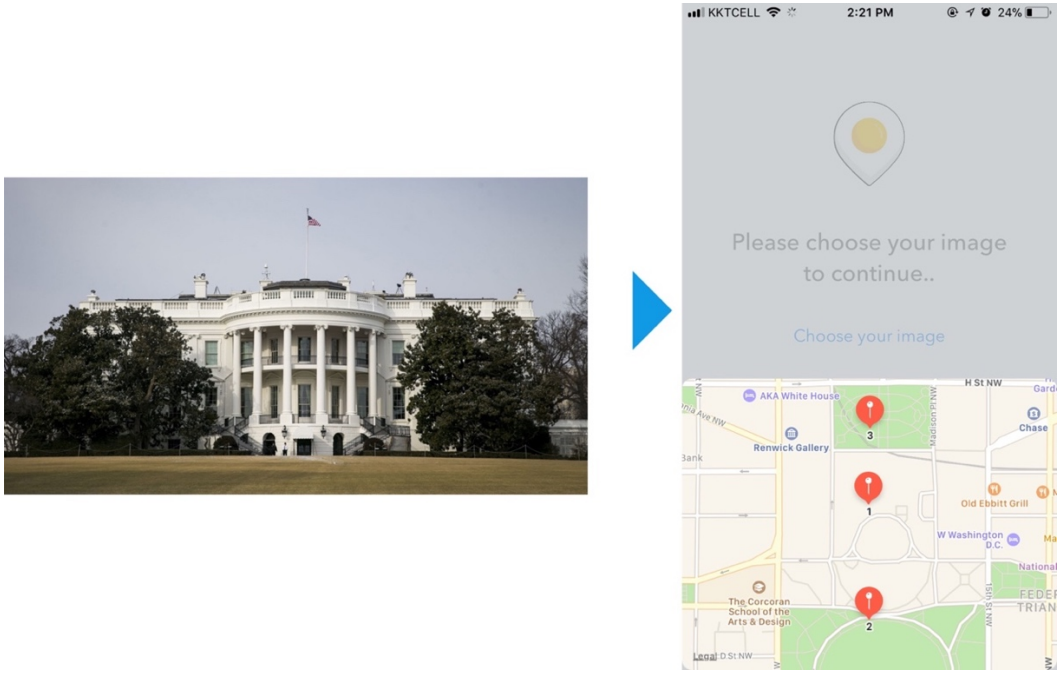


Figure 7.5 Mobile Application predicting White House located in Washington DC, USA

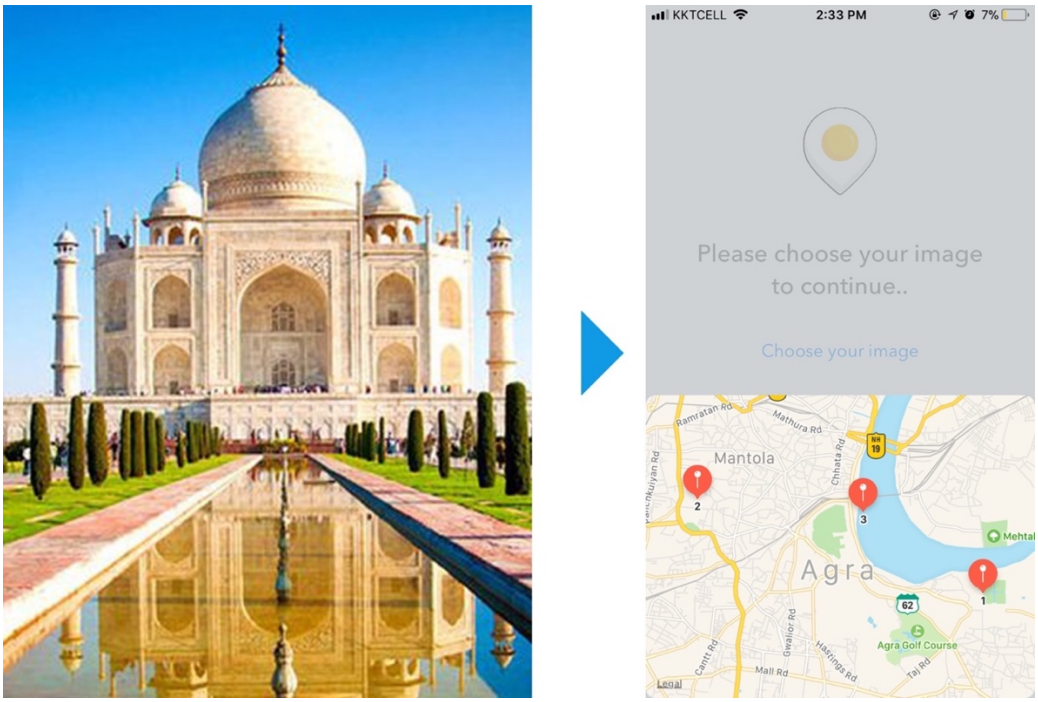


Figure 7.6 Mobile Application predicting Taj Mahal located in Agra, India

7.3 Experimental Evaluation

For the Food-101 dataset, 750 images of each class are used for training and the remaining 250 for testing by using the following python command.

```
//for splitting the dataset into training and testing sets

X_test, y_test = load_images('food-101/test', min_side=299)
X_train, y_train = load_images('food-101/train', min_side=299)
```

Figure 7.7 Splitting the dataset into training and testing sets

We measure performance with average accuracy, i.e. the fraction of test images that are correctly classified. We first give details of our implementation and then analyze the robustness of our approach with respect to its different parameters.

7.3.1 Dataset from Kaggle

We downloaded the dataset food-101 from Kaggle and extracted it within the notebook folder. Food classes are represented in Table 7.1. Some random images are shown in Figure 7.7.

Table 7.1 Representation of Food 101 Classes

apple_pie	dumplings	macaroni_and_cheese	steak
baby_back_ribs	edamame	macarons	strawberry_shortcake
baklava	eggs_benedict	miso_soup	sushi
beef_carpaccio	escargots	mussels	tacos
beef_tartare	falafel	nachos	takoyaki
beet_salad	filet_mignon	omelette	tiramisu
beignets	fish_and_chips	onion_rings	tuna_tartare
bibimbap	foie_gras	oysters	waffles
bread_pudding	french_fries	pad_thai	
bruschetta	french_onion_soup	paella	
caesar_salad	french_toast	pancakes	
caprese_salad	fried_calamari	panna_cotta	
carrot_cake	fried_rice	peking_duck	
ceviche	frozen_yogurt	pho	
cheesecake	garlic_bread	pizza	
cheese_plate	gnocchi	pork_chop	
chicken_curry	greek_salad	poutine	
chicken_quesadilla	grilled_cheese_sandwich	prime_rib	
chicken_wings	grilled_salmon	pulled_pork_sandwich	
chocolate_cake	guacamole	ramen	
chocolate_mousse	gyoza	ravioli	
churros	hamburger	red_velvet_cake	
clam_chowder	hot_and_sour_soup	risotto	
club_sandwich	hot_dog	samosa	
crab_cakes	huevos_rancheros	sashimi	
creme_brulee	hummus	scallops	
croque_madame	ice_cream	seaweed_salad	
cup_cakes	lasagna	shrimp_and_grits	
deviled_eggs	lobster_bisque	spaghetti_carbonara	
donuts	lobster_roll_sandwich	spring_rolls	

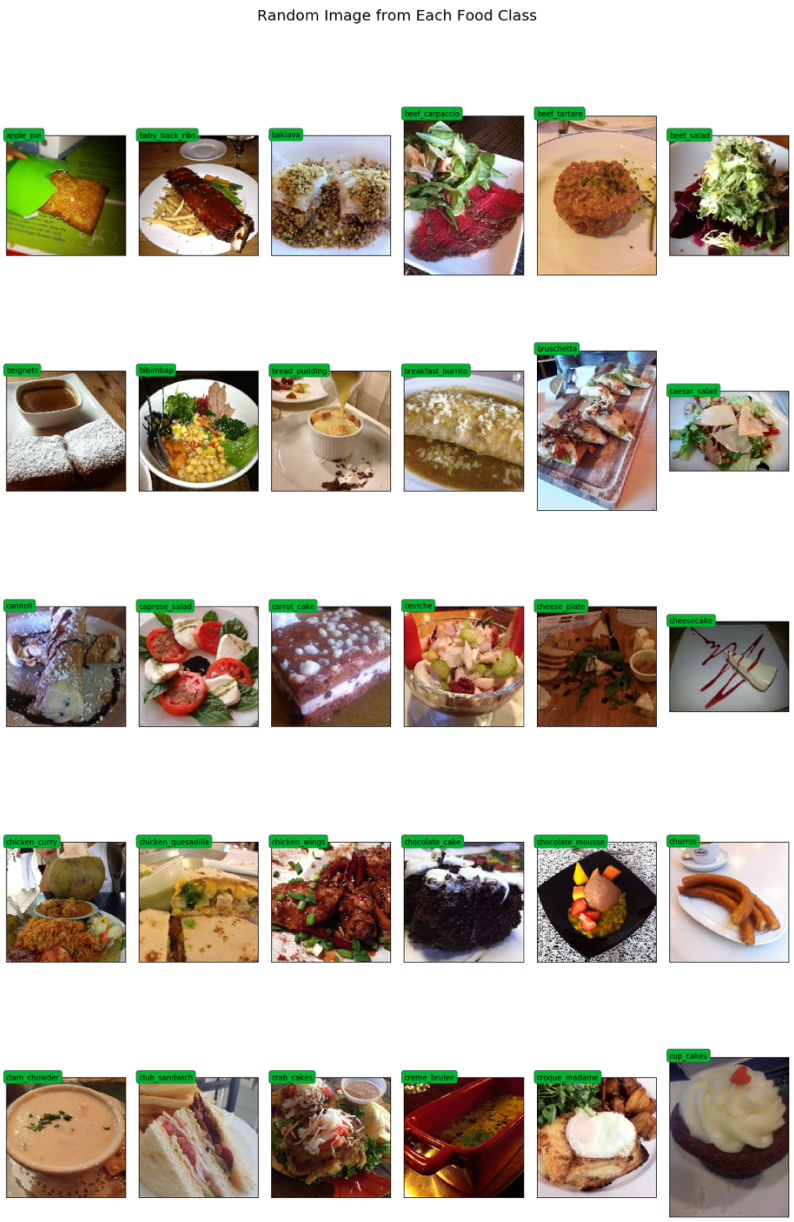


Figure 7.8 Examples of some random image from Each food class

7.3.2 Visualization Tools for Result Analysis

Here we have discussed some of the visualization tools that were helpful in visualizing the training and test dataset. We used python for visualizing the results. Here I am going through some visualizations. From Figure 7.8 – 7.11, code snippet and visualizations for training and testing datasets are shown.

```
@interact(n=(0, len(X_train)))
def show_pic(n):
    plt.imshow(X_train[n])
    print('class:', y_train[n], ix_to_class[y_train[n]])
```

Figure 7.9 Python Code for Visualizing Training Data Sets

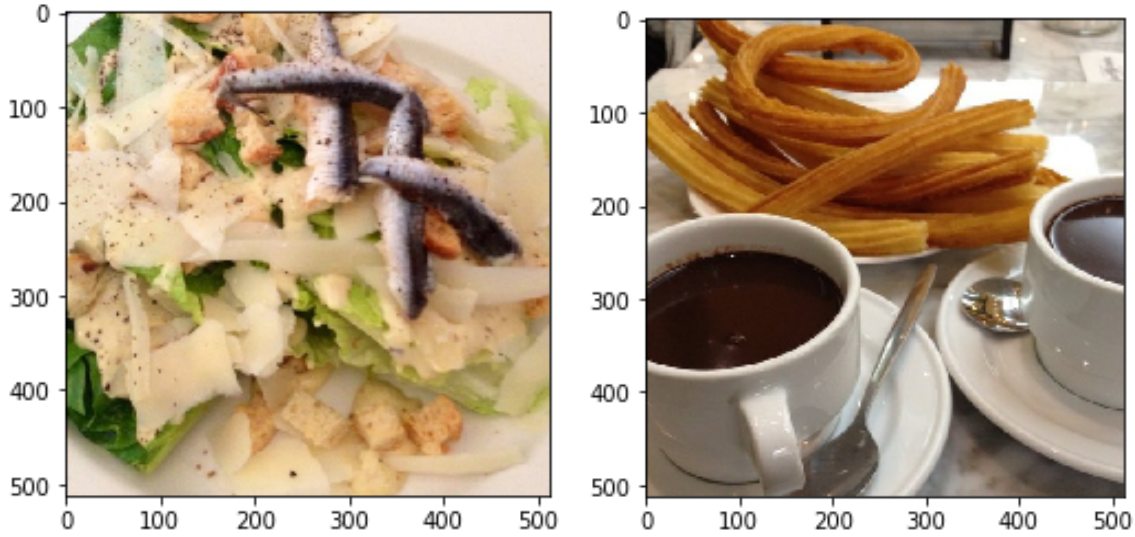


Figure 7.10 Visualizing random images from training dataset

```
@interact(n=(0, len(X_test)))
def show_pic(n):
    plt.imshow(X_test[n])
    print('class:', y_test[n], ix_to_class[y_test[n]])
```

Figure 7.11 Python Code for Visualizing Testing Data Sets

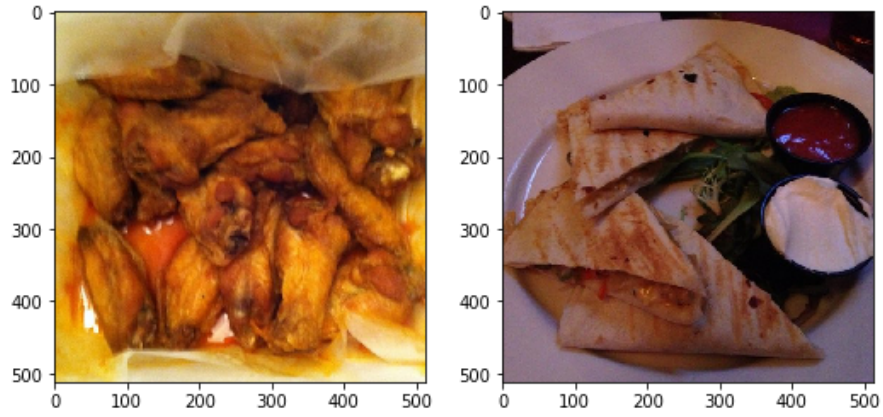


Figure 7.12 Visualizing Testing Data Sets

```
@interact(n_class=sorted_class_to_ix)
def show_random_images_of_class(n_class=0):
    print(n_class)
    nrows = 6
    ncols = 7
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols)
    fig.set_size_inches(12, 8)
    #fig.tight_layout()
    imgs = np.random.choice((y_train == n_class).nonzero()[0], nrows * ncols)
    for i, ax in enumerate(axes.flat):
        im = ax.imshow(X_train[imgs[i]])
        ax.set_axis_off()
        ax.title.set_visible(False)
        ax.xaxis.set_ticks([])
        ax.yaxis.set_ticks([])
        for spine in ax.spines.values():
            spine.set_visible(False)
    plt.subplots_adjust(left=0, wspace=0, hspace=0)
    plt.show()
```

Figure 7.13 Python Code for Visualizing some images of Baklawa having rows = 6 and columns = 7

Figure 7.14 Visualizing Baklava images from Class 21

7.3.3 Model Evaluation

At this point, several trained templates must be stored on the hard drive. We can browse it and use the load_model function to load the model with the lowest loss / precision. We also want to evaluate the set of tests with several crops. This can result in an increase in accuracy of 5% over a single crop. It is common to use the following crops: top left, top right, bottom left, bottom right, center. We also bring the same crops from left to right in the photo, for a total of 10 crops. We also want to return the N main forecasts for each crop, for example, in order to calculate the maximum accuracy.

```
def center_crop(x, center_crop_size, **kwargs):
    centerw, centerh = x.shape[0]//2, x.shape[1]//2
    halfw, halfh = center_crop_size[0]//2, center_crop_size[1]//2
    return x[centerw-halfw:centerw+halfw+1,centerh-halfh:centerh+halfh+1, :]
def predict_10_crop(img, ix, top_n=5, plot=False, preprocess=True, debug=False):
    flipped_X = np.fliplr(img)
    crops = [
        img[:299,:299, :], # Upper Left
```

```
        img[:299, img.shape[1]-299:, :], # Upper Right
        img[img.shape[0]-299:, :299, :], # Lower Left
        img[img.shape[0]-299:, img.shape[1]-299:, :], # Lower Right
        center_crop(img, (299, 299)),
    ]
    if preprocess:
        crops = [preprocess_input(x.astype('float32')) for x in crops]

    if plot:
        fig, ax = plt.subplots(2, 5, figsize=(10, 4))
        ax[0][0].imshow(crops[0])
        ax[0][1].imshow(crops[1])
        ax[0][2].imshow(crops[2])
        ax[0][3].imshow(crops[3])
        ax[0][4].imshow(crops[4])
        ax[1][0].imshow(crops[5])
        ax[1][1].imshow(crops[6])
        ax[1][2].imshow(crops[7])
        ax[1][3].imshow(crops[8])
        ax[1][4].imshow(crops[9])

    y_pred = model.predict(np.array(crops))
    preds = np.argmax(y_pred, axis=1)
    top_n_preds= np.argpartition(y_pred, -top_n)[:,-top_n:]
    return preds, top_n_preds
ix = 1300
predict_10_crop(X_test[ix], ix, top_n=5, plot=True, preprocess=False, debug=True)
```

Figure 7.15 Crop Code for Model Evaluation

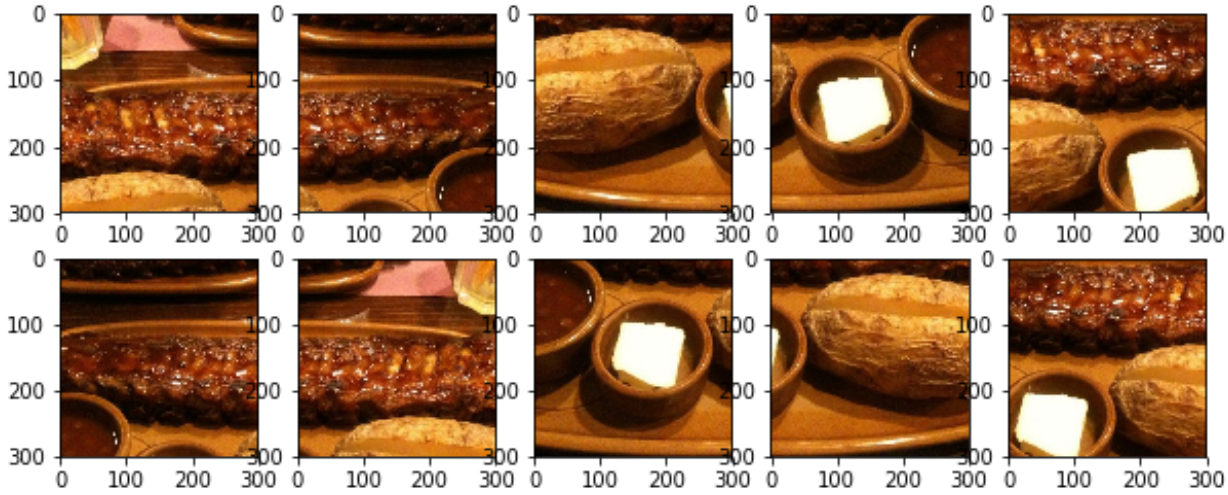


Figure 7.16 Multiple crops of a single image for inception model

We also need to preprocess the images for the Inception model

```
ix = 1300
predict_10_crop(X_test[ix], ix, top_n=5, plot=True, preprocess=True, debug=True)
```

Figure 7.17 Image preprocessing Code

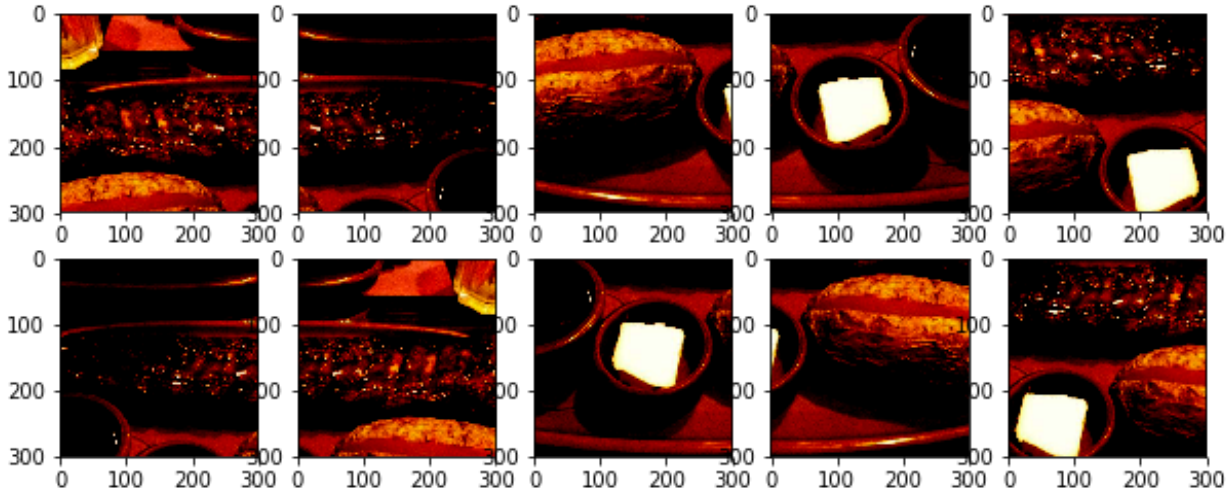


Figure 7.18 Image preprocessing for inception model

Now we will crop each image of the test set and get the predictions. It is a slow process because we are not using multiprocessors or other types of parallelism. We have trained the model on a MacBook which has core i7 processors. After the preprocessing of the cropped image we have now 10 predictions for each image which will boost up the accuracy of the model by 5%.

CHAPTER 8
CONCLUSION AND FUTURE WORK

In this work, an IOS based mobile application is developed based on CNN Machine Learning model. We divided our work into two parts. First part is based on food prediction. In this section, we created a machine learning model using CNN based inception V3 model as its top-5 error rate was very low. Then we compared the Inception V3 model with models such as AlexNet, Inception (GoogleNet) and BN-Inception V2 model. Results show that we can achieve an accuracy of 97.00% for our food prediction model. Second section is the location prediction model. We took this model from MIT places. Using Caffe model, we transformed the machine learning models to CoreML, which is the apple machine learning framework. Using swift programming language in Xcode IDE we developed the application and integrated both food and location models. The mobile application will allow users to capture food or location-based image and will input to IOS mobile application. If a user clicks location prediction then the application will show three best matching spots on Google maps using Google Maps API. In case of food prediction, there is no need of internet and once the food image is uploaded to mobile application, our mobile application will load and scan the image and will show the name of food with percentage of accuracy.

We compared our model with other machine learning models and found that in past research no food-based machine learning model has used in IOS mobile application and also the size of their models was much greater than ours. Our model size is 86 MB for food prediction and 150 MB for location-based model. Also, the computational cost of our machine learning model is less than that of others. We trained our machine learning model on a normal MacBook and then the model was trained in 6 to 7 hours. So, our neural network and model is much more cost effective and accurate.

We can improve the model by increasing its Top-1 and Top-3 accuracy by increasing the size of Epochs and also by increasing the inception blocks in Inception V3 model. Right now, we are having 32 Epochs and the error rate for Top - 1 is 18 percent while for Top – 3 is 13 percent. Also, by training the model using more data can increase the accuracy of the model.

In future we would like to combine the location and food predictions and will make it in such a way that when a user scans a food image and after prediction the application will also show the nearby restaurants having that food. Also, if a user scans a location-based image the application will show related restaurants nearby. In future we would add more inception blocks and more epochs to increase the accuracy of the model. Right now, the size of application is 300 MB so using cloud computing techniques we will train the models on google cloud and will extract the data using mobile application. By doing this the size of the application will be reduced and will be easily used by much more users. We will try to use more layers beside inception v3 layers and will compare the results. We might use a new layer “residual layer” which can eliminate any residual and non-important data present in the model. By using this layer, the size of the model will get reduce while its accuracy will be same. So, it will then also take less time in training the model.

REFERENCES

Liu, X., Datta, A., & Lim, E.-P. (2014). *Computational trust models and machine learning*. Boca Raton: CRC Press/Taylor & Francis Group.

Turing, A. M., & Yang, X.-S. (2013). *Artificial intelligence, evolutionary computing and metaheuristics : in the footsteps of Alan Turing*. Heidelberg: Springer.

Sharma, S. K., Al-Badi, A., Rana, N. P., & Al-Azizi, L. (2018). Mobile applications in government services (mG-App) from user's perspectives: A predictive modelling approach. *Government Information Quarterly, 35*(4), 557-568.

Hope, T., Resheff, Y. S., & Lieder, I. (2017). *Learning TensorFlow: a guide to building deep learning systems* (First edition. ed.). Sebastopol, CA: O'Reilly Media.

Müller, A. C., & Guido, S. (2016). *Introduction to machine learning with Python: a guide for data scientists* (First edition. ed.). Sebastopol, CA: O'Reilly Media, Inc.

Suskie, L. A., & American Association for Higher Education. (2001). *Assessment to promote deep learning*. Washington, DC: American Association for Higher Education.

Institute of Electrical and Electronics Engineers, & Computational Intelligence Society. (2013). *2013 IEEE Workshop on Robotic Intelligence in Informationally Structured Space (RiiSS 2013) Singapore, 16-19 April 2013; [part of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI)]*. Piscataway, NJ: IEEE.

I Venkatesan, R., & Li, B. (2018). *Convolutional neural networks in visual computing: a concise guide*. Boca Raton: CRC Press, Taylor & Francis Group, CRC Press is an imprint of the Taylor & Francis Group.

Keur, C., & Hillegass, A. (2016). *iOS programming: The Big Nerd Ranch guide* (Sixth edition. / ed.). Atlanta, GA: Big Nerd Ranch.

Yang, S., Yamauchi, K., Nonokawa, M., & Ikeda, M. (2002). Use of an artificial neural network to differentiate between ECGs with IRBBB patterns of atrial septal defect and healthy subjects. *Medical Informatics & the Internet in Medicine, 27*(1), 49-58.

Ventling, J. (2012). Fizzy's Lunch Lab: Fresh Food 101. *School Library Journal, 58*(3), 72-73.

Zuo, Z., Shuai, B., Wang, G., Liu, X., Wang, X., Wang, B., & Chen, Y. (2016). Learning Contextual Dependence with Convolutional Hierarchical Recurrent Neural Networks. *IEEE Trans Image Process, 25*(7), 2983-2996. doi:10.1109/TIP.2016.2548241

Tang, P., Wang, X., Shi, B., Bai, X., Liu, W., & Tu, Z. (2018). Deep FisherNet for Image Classification. *IEEE Trans Neural Netw Learn Syst*. doi:10.1109/TNNLS.2018.2874657

Dehmer, M., Mowshowitz, A., & Emmert-Streib, F. (2013). *Advances in network complexity*. Weinheim: Wiley-Blackwell.

Zheng, J., Cao, X., Zhang, B., Zhen, X., & Su, X. (2018). Deep Ensemble Machine for Video Classification. *IEEE Trans Neural Netw Learn Syst*. doi:10.1109/TNNLS.2018.2844464

Diaz, P., Montero, S., & Aedo, I. (2005). Modelling hypermedia and web applications: the Ariadne Development Method. *Information Systems, 30*(8), 649-673.

Wei, Y., Zhao, Y., Lu, C., Wei, S., Liu, L., Zhu, Z., & Yan, S. (2017). Cross-Modal Retrieval with CNN Visual Features: A New Baseline. *IEEE Trans Cybern, 47*(2), 449-460. doi:10.1109/TCYB.2016.2519449

Shaw, B. I., Wangara, A. A., Wambua, G. M., Kiruja, J., Dicker, R. A., Mweu, J. M., & Juillard, C. (2017). Geospatial relationship of road traffic crashes and healthcare facilities with

trauma surgical capabilities in Nairobi, Kenya: defining gaps in coverage. *Trauma Surg Acute Care Open, 2*(1), e000130. doi:10.1136/tsaco-2017-000130

Ramampiaro, H., Langseth, H., Agarwal, B., & Ruocco, M. (2018). A deep network model for paraphrase detection in short text messages. *Information Processing & Management, 54*(6), 922-937.

Hwang, K., & Chen, M. (2017). *Big-Data Analytics for Cloud, IoT and Cognitive Computing*. Chichester, UK; Hoboken, NJ: John Wiley & Sons.

Ramsundar, B., & Zadeh, R. B. (2018). *TensorFlow for deep learning: from linear regression to reinforcement learning* (First edition. ed.). Beijing: O'Reilly Media.

Kar, A., Bera, S., Karri, S. P. K., Ghosh, S., Mahadevappa, M., & Sheet, D. (2018). A Deep Convolutional Neural Network Based Classification of Multi-Class Motor Imagery with Improved Generalization. *Conf Proc IEEE Eng Med Biol Soc, 2018*, 5085-5088. doi:10.1109/EMBC.2018.8513451

Tayara, H., & Chong, K. T. (2018). Object Detection in Very High-Resolution Aerial Images Using One-Stage Densely Connected Feature Pyramid Network. *Sensors (Basel), 18*(10). doi:10.3390/s18103341

An, Z., Xu, X., Yang, J., Liu, Y., & Yan, Y. (2018). Research of the three-dimensional tracking and registration method based on multiobjective constraints in an AR system. *Appl Opt, 57*(32), 9625-9634. doi:10.1364/AO.57.009625

APPENDICES

APPENDIX 1

MACHINE LEARNING MODEL

Loading and Preprocessing Dataset

```
import matplotlib.pyplot as plt
import matplotlib.image as img
import numpy as np
from scipy.misc import imresize
import os
from os import listdir
from os.path import isfile, join
import shutil
import stat
import collections
from collections import defaultdict
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets
import h5py
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.applications.inception_v3 import preprocess_input
from keras.models import load_model
```

Importing all of the packages for Model Training

Let's look at some random images from each food class.

```
root_dir = 'owais_food_dataset'
rows = 5
cols = 6
fig, ax = plt.subplots(rows, cols, frameon=False, figsize=(15, 25))
fig.suptitle('Random Image from Each Food Class', fontsize=20)
sorted_food_dirs = sorted(os.listdir(root_dir))
for i in range(rows):
    for j in range(cols):
        try:
            food_dir = sorted_food_dirs[i*cols + j]
        except:
            break
        all_files = os.listdir(os.path.join(root_dir, food_dir))
        rand_img = np.random.choice(all_files)
```

```
        img = plt.imread(os.path.join(root_dir, food_dir, rand_img))
        ax[i][j].imshow(img)
        ec = (0, .6, .1)
        fc = (0, .7, .2)
        ax[i][j].text(0, -20, food_dir, size=10, rotation=0,
                ha="left", va="top",
                bbox=dict(boxstyle="round", ec=ec, fc=fc))
plt.setp(ax, xticks=[], yticks=[])
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

Code for Selecting Random Food Images

Multiprocessing Pool was used for accelerating image augmentation during training session.

```
import multiprocessing as mp
num_processes = 6
pool = mp.Pool(processes=num_processes)
```

Multiprocessing Pool

```
def load_images(root, min_side=299):
    all_imgs = []
    all_classes = []
    resize_count = 0
    invalid_count = 0
    for i, subdir in enumerate(listdir(root)):
        imgs = listdir(join(root, subdir))
        class_ix = class_to_ix[subdir]
        print(i, class_ix, subdir)
        for img_name in imgs:
            img_arr = img.imread(join(root, subdir, img_name))
            img_arr_rs = img_arr
            try:
                w, h, _ = img_arr.shape
                if w < min_side:
                    wpercent = (min_side/float(w))
                    hsize = int((float(h)*float(wpercent)))
                    #print('new dims:', min_side, hsize)
                    img_arr_rs = imresize(img_arr, (min_side, hsize))
                    resize_count += 1
                elif h < min_side:
                    hpercent = (min_side/float(h))
                    wsize = int((float(w)*float(hpercent)))
```

```
                    #print('new dims:', wsize, min_side)
                    img_arr_rs = imresize(img_arr, (wsize, min_side))
                    resize_count += 1
                all_imgs.append(img_arr_rs)
                all_classes.append(class_ix)
            except:
                print('Skipping bad image: ', subdir, img_name)
                invalid_count += 1
    print(len(all_imgs), 'images loaded')
    print(resize_count, 'images resized')
    print(invalid_count, 'images skipped')
    return np.array(all_imgs), np.array(all_classes)
X_test, y_test = load_images('owais_food_dataset', min_side=299)
```

Load dataset images and resize them to meet minimum width and height according to pixel size

APPENDIX 2

VISUALIZATION TOOLS

```
@interact(n=(0, len(X_train)))
def show_pic(n):
    plt.imshow(X_train[n])
    print('class:', y_train[n], ix_to_class[y_train[n]])
```

Visualizing a random image from training folder

```
@interact(n=(0, len(X_test)))
def show_pic(n):
    plt.imshow(X_test[n])
    print('class:', y_test[n], ix_to_class[y_test[n]])
```

Visualizing a random image from testing folder

```
@interact(n_class=sorted_class_to_ix)
def show_random_images_of_class(n_class=0):
    print(n_class)
    nrows = 4
    ncols = 8
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols)
    fig.set_size_inches(12, 8)
    #fig.tight_layout()
    imgs = np.random.choice((y_train == n_class).nonzero()[0], nrows *
ncols)
    for i, ax in enumerate(axes.flat):
        im = ax.imshow(X_train[imgs[i]])
        ax.set_axis_off()
        ax.title.set_visible(False)
        ax.xaxis.set_ticks([])
        ax.yaxis.set_ticks([])
        for spine in ax.spines.values():
            spine.set_visible(False)
    plt.subplots_adjust(left=0, wspace=0, hspace=0)
    plt.show()
```

Visualizing some images of a food class having rows = 4 and columns = 8

```
@interact(n_class=sorted_class_to_ix)
def show_random_images_of_class(n_class=0):
    print(n_class)
    nrows = 4
```

```
    ncols = 8
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols)
    fig.set_size_inches(12, 8)
    #fig.tight_layout()
    imgs = np.random.choice((y_test == n_class).nonzero()[0], nrows *
ncols)
    for i, ax in enumerate(axes.flat):
        im = ax.imshow(X_test[imgs[i]])
        ax.set_axis_off()
        ax.title.set_visible(False)
        ax.xaxis.set_ticks([])
        ax.yaxis.set_ticks([])
        for spine in ax.spines.values():
            spine.set_visible(False)
    plt.subplots_adjust(left=0, wspace=0, hspace=0)
    plt.show()
```

Appling Randomization again to a food Class for Verification

```
from keras.utils.np_utils import to_categorical

n_classes = 101
y_train_cat = to_categorical(y_train, nb_classes=n_classes)
y_test_cat = to_categorical(y_test, nb_classes=n_classes)
In [18]:
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input,
decode_predictions
from keras.preprocessing import image
from keras.layers import Input

import tools.image_gen_extended as T

# Useful for checking the output of the generators after code change
#from importlib import reload
#reload(T)
```

Image Augmentation

APPENDIX 3

MODEL EVALUATION

```
def center_crop(x, center_crop_size, **kwargs):
    centerw, centerh = x.shape[0]//2, x.shape[1]//2
    halfw, halfh = center_crop_size[0]//2, center_crop_size[1]//2
    return x[centerw-halfw:centerw+halfw+1,centerh-
halfh:centerh+halfh+1, :]
def predict_10_crop(img, ix, top_n=5, plot=False, preprocess=True,
debug=False):
    flipped_X = np.fliplr(img)
    crops = [
        img[:299,:299, :], # Upper Left
        img[:299, img.shape[1]-299:, :], # Upper Right
        img[img.shape[0]-299:, :299, :], # Lower Left
        img[img.shape[0]-299:, img.shape[1]-299:, :], # Lower Right
        center_crop(img, (299, 299)),

        flipped_X[:299,:299, :],
        flipped_X[:299, flipped_X.shape[1]-299:, :],
        flipped_X[flipped_X.shape[0]-299:, :299, :],
        flipped_X[flipped_X.shape[0]-299:, flipped_X.shape[1]-299:,
:],
        center_crop(flipped_X, (299, 299))
    ]
    if preprocess:
        crops = [preprocess_input(x.astype('float32')) for x in crops]

    if plot:
        fig, ax = plt.subplots(2, 5, figsize=(10, 4))
        ax[0][0].imshow(crops[0])
        ax[0][1].imshow(crops[1])
        ax[0][2].imshow(crops[2])
        ax[0][3].imshow(crops[3])
        ax[0][4].imshow(crops[4])
        ax[1][0].imshow(crops[5])
        ax[1][1].imshow(crops[6])
        ax[1][2].imshow(crops[7])
        ax[1][3].imshow(crops[8])
        ax[1][4].imshow(crops[9])

    y_pred = model.predict(np.array(crops))
    preds = np.argmax(y_pred, axis=1)
    top_n_preds= np.argpartition(y_pred, -top_n)[:,-top_n:]
    if debug:
```

```
        print('Top-1 Predicted:', preds)
        print('Top-5 Predicted:', top_n_preds)
        print('True Label:', y_test[ix])
    return preds, top_n_preds


ix = 13001
predict_10_crop(X_test[ix], ix, top_n=5, plot=True, preprocess=False,
debug=True)
```

Model Evaluation

```
%%time
preds_10_crop = {}
for ix in range(len(X_test)):
    if ix % 1000 == 0:
        print(ix)
    preds_10_crop[ix] = predict_10_crop(X_test[ix], ix)
preds_uniq = {k: np.unique(v[0]) for k, v in preds_10_crop.items()}
preds_hist = np.array([len(x) for x in preds_uniq.values()])

plt.hist(preds_hist, bins=11)
plt.title('Number of unique predictions per image')
```

Number of unique predictions per image

```
preds_top_1 = {k: collections.Counter(v[0]).most_common(1) for k, v in
preds_10_crop.items()}

top_5_per_ix = {k: collections.Counter(preds_10_crop[k][1].reshape(-
1)).most_common(5)
                for k, v in preds_10_crop.items()}
preds_top_5 = {k: [y[0] for y in v] for k, v in top_5_per_ix.items()}
%%time
right_counter = 0
for i in range(len(y_test)):
    guess, actual = preds_top_1[i][0][0], y_test[i]
    if guess == actual:
        right_counter += 1

print('Top-1 Accuracy, 10-Crop: {0:.2f}%'.format(right_counter /
len(y_test) * 100))

Top-1 Accuracy, 10-Crop: 86.97%
CPU times: user 28 ms, sys: 0 ns, total: 28 ms
```

```
Wall time: 27.3 ms

In [134]:
%%time
top_5_counter = 0
for i in range(len(y_test)):
    guesses, actual = preds_top_5[i], y_test[i]
    if actual in guesses:
        top_5_counter += 1

print('Top-5 Accuracy, 10-Crop: {0:.2f}%'.format(top_5_counter /
len(y_test) * 100))
```

Results Visualization

APPENDIX 4

XCODE CODE

First View Controller

```
import UIKit
import Photos
import GoogleMobileAds

class FirstViewController: UIViewController {

    @IBOutlet var selectImage: UIButton!
    var interstitial: GADInterstitial?
    var interstialAdUnitID : String = "Your_ID"
    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    @IBAction func chooseYourImage(_ sender: Any) {
        var libraryEnabled: Bool = true
        var croppingEnabled: Bool = false
        var allowResizing: Bool = true
        var allowMoving: Bool = false
        var minimumSize: CGSize = CGSize(width: 60, height: 60)

        var croppingParameters: CroppingParameters {
            return CroppingParameters(isEnabled:
croppingEnabled, allowResizing: allowResizing, allowMoving:
allowMoving, minimumSize: minimumSize)
        }

        let alert = UIAlertController(title: "Choose an image",
message: "", preferredStyle: UIAlertControllerStyle.actionSheet)
        alert.addAction(UIAlertAction(title: "Cancel", style:
UIAlertActionStyle.cancel, handler: nil))
        alert.addAction(UIAlertAction(title: "Open Camera",
style: UIAlertActionStyle.default, handler: { action in
```

```
            //let croppingEnabled = true
            let cameraViewController =
CameraViewController(croppingParameters: croppingParameters,
allowsLibraryAccess: libraryEnabled) { [weak self] image, asset
in
                let userdef = UserDefaults.standard
                let images = UIImageJPEGRepresentation(image!,
1)
                userdef.set(images, forKey: "selectedimage")
                userdef.synchronize()


                self?.dismiss(animated: true, completion: nil)

                self?.performSegue(withIdentifier: "toOptions",
sender: self)
                self?.dismiss(animated: true, completion: nil)
            }
            if let presenter =
alert.popoverPresentationController {
                presenter.sourceView = self.selectImage
                presenter.sourceRect = self.selectImage.bounds
            }
           self.present(cameraViewController, animated: true,
completion: nil)

        }))
        alert.addAction(UIAlertAction(title: "Photo Library",
style: UIAlertActionStyle.default, handler: { action in
            let imagePickerViewController =
PhotoLibraryViewController()
            imagePickerViewController.onSelectionComplete = {
asset in

                // Provides a PHAsset object
                // Retrieve a UIImage from a PHAsset using
                let options = PHImageRequestOptions()
                options.deliveryMode = .highQualityFormat
                options.isNetworkAccessAllowed = true

                PHImageManager.default().requestImage(for:
asset!, targetSize: PHImageManagerMaximumSize, contentMode:
.aspectFill, options: options) { image, _ in
                    if let image = image {
                        let userdef = UserDefaults.standard
```

```
                        let images1 =
UIImageJPEGRepresentation(image, 1)
                        userdef.set(images1, forKey:
"selectedimage")
                        userdef.synchronize()


                        self.dismiss(animated: true, completion:
nil)

                        let vc =
self.storyboard?.instantiateViewController(withIdentifier:
"option") as! OptionsViewController

                        self.present(vc, animated: false,
completion: nil)
                    }
                }
            }

            self.present(imagePickerViewController, animated:
true, completion: nil)
        }))

        alert.popoverPresentationController?.sourceView =
self.selectImage // works for both iPhone & iPad

        self.present(alert, animated: true) {
            print("option menu presented")
        }
       // self.present(alert, animated: true, completion: nil)


    }


    private func createAndLoadInterstitial() -> GADInterstitial?
{
        interstitial = GADInterstitial(adUnitID:
interstialAdUnitID)

        guard let interstitial = interstitial else {
            return nil
        }
```

```
        let request = GADRequest()
        // Remove the following line before you upload the app
        // request.testDevices = [ kGADSimulatorID ]
        interstitial.load(request)
        interstitial.delegate = self as! GADInterstitialDelegate

        return interstitial

    }

    func interstitialDidReceiveAd(_ ad: GADInterstitial) {
        print("Interstitial loaded successfully")


        ad.present(fromRootViewController: self)
    }

    func interstitialDidFail(toPresentScreen ad:
GADInterstitial) {
        print("Fail to receive interstitial")
    }
}
```

Options View Controller

```
import UIKit
import AMPopTip

class OptionsViewController: UIViewController {

    @IBOutlet var mapImage: UIImageView!
    @IBOutlet var foodImage: UIImageView!
    @IBOutlet var secondView: SpringView!
    override func viewDidLoad() {
        super.viewDidLoad()
        let tapGestureRecognizer =
UITapGestureRecognizer(target: self, action:
#selector(imageTapped(tapGestureRecognizer:)))
        mapImage.isUserInteractionEnabled = true
        mapImage.addGestureRecognizer(tapGestureRecognizer)

        let tapGestureRecognizer1 =
UITapGestureRecognizer(target: self, action:
#selector(imageTapped1(tapGestureRecognizer:)))
        foodImage.isUserInteractionEnabled = true
        foodImage.addGestureRecognizer(tapGestureRecognizer1)
```

```
        let swipeDown = UISwipeGestureRecognizer(target: self,
action: #selector(respondToSwipeGesture))
        swipeDown.direction =
UISwipeGestureRecognizerDirection.down
        self.view.addGestureRecognizer(swipeDown)

        let ud = UserDefaults.standard


        let bool =  ud.bool(forKey: "tipSearch")
        if !bool {
            let popTip = PopTip()
            popTip.show(text: "Swipe me down to dismiss",
direction: .up, maxWidth: 200, in: view, from: secondView.frame)
            popTip.shouldDismissOnTap = true

            ud.set(true, forKey: "tipSearch")
            ud.synchronize()
        } else {

        }

    }

    @objc func respondToSwipeGesture(gesture:
UIGestureRecognizer) {
        if let swipeGesture = gesture as?
UISwipeGestureRecognizer {
            switch swipeGesture.direction {
            case UISwipeGestureRecognizerDirection.right:
                print("Swiped right")
            case UISwipeGestureRecognizerDirection.down:
                print("Swiped down")
                self.dismiss(animated: true, completion: nil)

            case UISwipeGestureRecognizerDirection.left:
                print("Swiped left")
            case UISwipeGestureRecognizerDirection.up:
                print("Swiped up")
            default:
                break
            }
        }
    }
```

```
    @objc func imageTapped(tapGestureRecognizer:
UITapGestureRecognizer)
    {

        SwiftSpinner.show("Scanning your image..")
        DispatchQueue.main.asyncAfter(deadline: .now() + 3.0) {
            let vc =
self.storyboard?.instantiateViewController(withIdentifier:
"map") as! MapViewController

            self.present(vc, animated: false, completion: nil)
            SwiftSpinner.hide({
                //do stuff
            })
        }



        //self.performSegue(withIdentifier: "toMap", sender:
self)

    }
    @objc func imageTapped1(tapGestureRecognizer:
UITapGestureRecognizer)
    {

        SwiftSpinner.show("Scanning your image...")
        DispatchQueue.main.asyncAfter(deadline: .now() + 3.0) {
            let vc =
self.storyboard?.instantiateViewController(withIdentifier:
"food") as! FoodViewController

            self.present(vc, animated: false, completion: nil)
            SwiftSpinner.hide({
                //do stuff
            })


    }


}
```

Map View Controller

```
import UIKit
import MapKit
import Vision
import MapKit
import Alamofire


class PredictionLocation: NSObject, MKAnnotation{
    var identifier = "Prediction location"
    var title: String?
    var coordinate: CLLocationCoordinate2D

init(name:String,lat:CLLocationDegrees,long:CLLocationDegrees){
        title = name
        coordinate = CLLocationCoordinate2DMake(lat, long)
    }
}

class PredictionLocationList: NSObject {
    var place = [PredictionLocation]()
    override init(){
        place += [PredictionLocation(name:"1",lat: 0, long: 0)]
        place += [PredictionLocation(name:"2",lat: 1, long: 1)]
        place += [PredictionLocation(name:"3",lat: 2, long: 2)]
    }
}
class MapViewController: UIViewController {
    // Define Core ML model
    // Make sure to add the file  in the Project Navigator, and
have Target Membership checked
    let model = RN1015k500()


    //MARK: - Map setup
    func resetRegion(){
        let region =
MKCoordinateRegionMakeWithDistance(annotation.coordinate, 5000,
5000)
        mapKit.setRegion(region, animated: true)
    }

    var myLatitude = ""
    var myLongitude = ""
```

```
    // Array of annotations
    let annotation = MKPointAnnotation()
    var places = PredictionLocationList().place
    var locationsArray = [String]()
    var ie: Int = 0
    @IBOutlet var mapKit: MKMapView!

    override func viewDidLoad() {
        super.viewDidLoad()

        let defaults = UserDefaults.standard
        let data = defaults.data(forKey: "selectedimage")
        let uiimage2 = UIImage(data: data!)
        defaults.synchronize()
        let image = uiimage2
        // imageView.image = image

        predictUsingVision(image: image!)

        let swipeDown = UISwipeGestureRecognizer(target: self,
action: #selector(respondToSwipeGesture))
        swipeDown.direction =
UISwipeGestureRecognizerDirection.down
        self.view.addGestureRecognizer(swipeDown)
        // Do any additional setup after loading the view.
    }
    @objc func respondToSwipeGesture(gesture:
UIGestureRecognizer) {
        if let swipeGesture = gesture as?
UISwipeGestureRecognizer {
            switch swipeGesture.direction {
            case UISwipeGestureRecognizerDirection.right:
                print("Swiped right")
            case UISwipeGestureRecognizerDirection.down:
                print("Swiped down")
                self.dismiss(animated: true, completion: nil)

            case UISwipeGestureRecognizerDirection.left:
                print("Swiped left")
            case UISwipeGestureRecognizerDirection.up:
                print("Swiped up")
            default:
                break
            }
        }
    }
```

```
    func predictUsingVision(image: UIImage) {
        guard let visionModel = try? VNCoreMLModel(for:
model.model) else {
            fatalError("Something went wrong")
        }

        let request = VNCoreMLRequest(model: visionModel) {
request, error in
            if let observations = request.results as?
[VNClassificationObservation] {
                let top3 = observations.prefix(through: 2)
                    .map { ($0.identifier,
Double($0.confidence)) }
                self.show(results: top3)
            }
        }

        request.imageCropAndScaleOption = .centerCrop

        let handler = VNImageRequestHandler(cgImage:
image.cgImage!)
        try? handler.perform([request])
    }

    typealias Prediction = (String, Double)

    func show(results: [Prediction]) {
        var s: [String] = []
        for (i, pred) in results.enumerated() {
            let latLongArr = pred.0.components(separatedBy:
"\t")
            print("lat long \(latLongArr)")
            myLatitude = latLongArr[1]
            myLongitude = latLongArr[2]

            ie = i

            s.append(String(format: "%d: %@ %@ (%.2f%%)", i + 1,
myLatitude, myLongitude, pred.1 * 100))
            LocationByCoordinates(latitude: myLatitude,
longitude: myLongitude)
```

```
            print("first latidue \(myLatitude),,,,
\(myLongitude)")

            places[i].title = String(i+1)
            places[i].coordinate =
CLLocationCoordinate2D(latitude: CLLocationDegrees(myLatitude)!,
longitude: CLLocationDegrees(myLongitude)!)


        }
       // predictionLabel.text = s.joined(separator: "\n")



        // Map reset
        resetRegion()

        mapKit.centerCoordinate = places[0].coordinate
        // Show annotations for the predictions on the map

        mapKit.addAnnotations(places)
        // Zoom map to fit all annotations
        zoomMapFitAnnotations()
    }
    func zoomMapFitAnnotations() {
        var zoomRect = MKMapRectNull
        for annotation in mapKit.annotations {

            let annotationPoint =
MKMapPointForCoordinate(annotation.coordinate)
            let pointRect = MKMapRectMake(annotationPoint.x,
annotationPoint.y, 0, 0)
            if (MKMapRectIsNull(zoomRect)) {
                zoomRect = pointRect
            } else {
                zoomRect = MKMapRectUnion(zoomRect, pointRect)
            }
        }
        self.mapKit.setVisibleMapRect(zoomRect, edgePadding:
UIEdgeInsetsMake(50, 50, 50, 50), animated: true)
    }

    func LocationByCoordinates (latitude:
String,longitude:String) {
        let mapsKey = UserDefaults.standard.string(forKey:
"maps_key") ?? ""
```

```
Alamofire.request("https://maps.googleapis.com/maps/api/geocode/
json?latlng=\(latitude),\(longitude)&key=\(mapsKey)").responseJS
ON { response in
            if let json = response.result.value {
                let request = json as? NSDictionary
                if let id = request!["results"]
                {
                    // print(id)
                    let ide = id as? NSArray
                    let formatted_address = ide![0]
                    let fors = formatted_address as!
NSDictionary

                    //print(fors.value(forKey:
"formatted_address"))
                    let forss = fors.value(forKey:
"formatted_address")

                    self.locationsArray.append(forss as? String
?? "")

                    if self.ie == 0 {
                        self.places[0].identifier = (forss as?
String)!
                    } else if self.ie == 1 {
                        self.places[1].identifier = (forss as?
String)!
                    } else if self.ie == 2 {
                        self.places[2].identifier = (forss as?
String)!
                    }

                }
            }
        }
       }

extension Collection where Indices.Iterator.Element == Index {
    subscript (safe index: Index) -> Iterator.Element? {
        return indices.contains(index) ? self[index] : nil
    }
}
```

Food View Controller

```
import UIKit
import CoreML
```

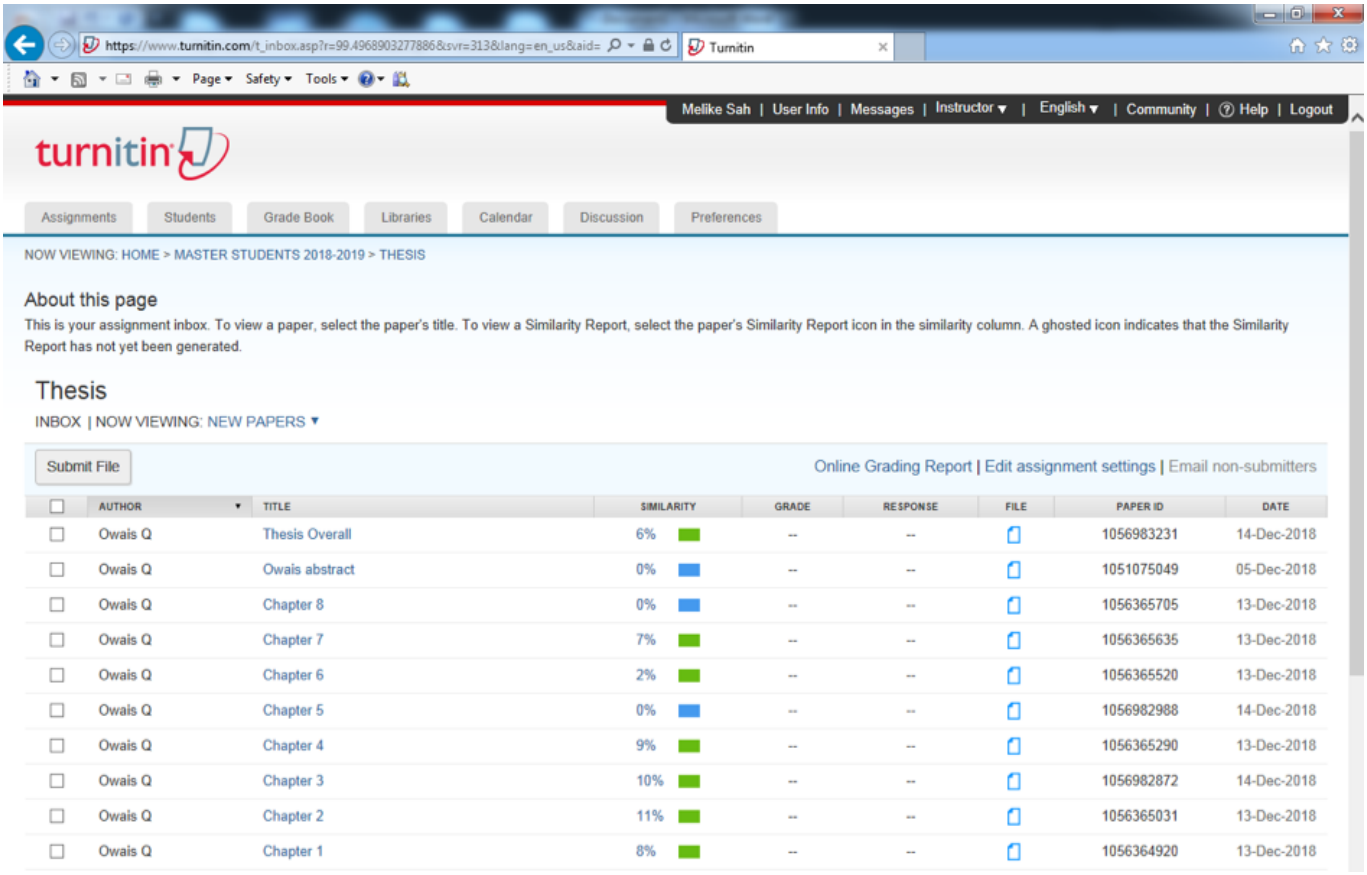
```
import MobileCoreServices
import Photos

extension UIImage {

    func resize(to newSize: CGSize) -> UIImage? {

        guard self.size != newSize else { return self }

        UIGraphicsBeginImageContextWithOptions(newSize, false,
0.0)
        self.draw(in: CGRect(x: 0, y: 0, width: newSize.width,
height: newSize.height))

        defer { UIGraphicsEndImageContext() }
        return UIGraphicsGetImageFromCurrentImageContext()
    }

    func pixelBuffer() -> CVPixelBuffer? {

        let width = Int(self.size.width)
        let height = Int(self.size.height)

        let attrs = [kCVPixelBufferCGImageCompatibilityKey:
kCFBooleanTrue, kCVPixelBufferCGBitmapContextCompatibilityKey:
kCFBooleanTrue] as CFDictionary
        var pixelBuffer: CVPixelBuffer?
        let status = CVPixelBufferCreate(kCFAllocatorDefault,
width, height, kCVPixelFormatType_32ARGB, attrs, &pixelBuffer)
        guard status == kCVReturnSuccess else {
            return nil
        }

        CVPixelBufferLockBaseAddress(pixelBuffer!,
CVPixelBufferLockFlags(rawValue: 0))
        let pixelData =
CVPixelBufferGetBaseAddress(pixelBuffer!)

        let rgbColorSpace = CGColorSpaceCreateDeviceRGB()
        let context = CGContext(data: pixelData, width: width,
height: height, bitsPerComponent: 8, bytesPerRow:
CVPixelBufferGetBytesPerRow(pixelBuffer!), space: rgbColorSpace,
bitmapInfo: CGImageAlphaInfo.noneSkipFirst.rawValue)

        context?.translateBy(x: 0, y: CGFloat(height))
        context?.scaleBy(x: 1.0, y: -1.0)
```

```
        UIGraphicsPushContext(context!)
        self.draw(in: CGRect(x: 0, y: 0, width: width, height:
height))
        UIGraphicsPopContext()
        CVPixelBufferUnlockBaseAddress(pixelBuffer!,
CVPixelBufferLockFlags(rawValue: 0))

        return pixelBuffer
    }
}

class FoodViewController: UIViewController {

    @IBOutlet var imageFood: UIImageView!

    @IBOutlet var name: UILabel!
    @IBOutlet var confidence: UILabel!
    override func viewDidLoad() {
        super.viewDidLoad()
        let defaults = UserDefaults.standard
        let data = defaults.data(forKey: "selectedimage")
        let uiimage2 = UIImage(data: data!)
        defaults.synchronize()
        let image1 = uiimage2
        self.imageFood.image = image1
        let model = Food101()
        let size = CGSize(width: 299, height: 299)
        let image = image1!

        guard let buffer = image.resize(to: size)?.pixelBuffer()
else {
            fatalError("Scaling or converting to pixel buffer
failed!")
        }

        guard let result = try? model.prediction(image: buffer)
else {
            fatalError("Prediction failed!")
        }

        let confidence =
result.foodConfidence["\(result.classLabel)"]! * 100.0
        let converted = String(format: "%.2f", confidence)
        let both = result.classLabel + " - " + converted + "%"
        self.name.text = both
```

```
        //self?.imageView.image = image
        // self?.percentage.text = "\(result.classLabel) -
\(converted) %"
        let swipeDown = UISwipeGestureRecognizer(target: self,
action: #selector(respondToSwipeGesture))
        swipeDown.direction =
UISwipeGestureRecognizerDirection.down
        self.view.addGestureRecognizer(swipeDown)

    }

    @objc func respondToSwipeGesture(gesture:
UIGestureRecognizer) {
        if let swipeGesture = gesture as?
UISwipeGestureRecognizer {
            switch swipeGesture.direction {
            case UISwipeGestureRecognizerDirection.right:
                print("Swiped right")
            case UISwipeGestureRecognizerDirection.down:
                print("Swiped down")
                self.dismiss(animated: true, completion: nil)

            case UISwipeGestureRecognizerDirection.left:
                print("Swiped left")
            case UISwipeGestureRecognizerDirection.up:
                print("Swiped up")
            default:
                break
            }
        }
    }

}
```

APPENDIX 5

TURNITIN REPORT



AUTHOR	TITLE	SIMILARITY	GRADE	RESPONSE	FILE	PAPER ID	DATE
Owais Q	Thesis Overall	6%	--	--		1056983231	14-Dec-2018
Owais Q	Owais abstract	0%	--	--		1051075049	05-Dec-2018
Owais Q	Chapter 8	0%	--	--		1056365705	13-Dec-2018
Owais Q	Chapter 7	7%	--	--		1056365635	13-Dec-2018
Owais Q	Chapter 6	2%	--	--		1056365520	13-Dec-2018
Owais Q	Chapter 5	0%	--	--		1056982988	14-Dec-2018
Owais Q	Chapter 4	9%	--	--		1056365290	13-Dec-2018
Owais Q	Chapter 3	10%	--	--		1056982872	14-Dec-2018
Owais Q	Chapter 2	11%	--	--		1056365031	13-Dec-2018
Owais Q	Chapter 1	8%	--	--		1056364920	13-Dec-2018