A Project Report

on

FOOD CALORIE PREDICTION USING DEEP LEARNING

Submitted in partial fulfillment of the

requirement for the award of the degree

of

Bachelor of Technology in Computer Science and Engineering

Under The Supervision of
Dr. Prashant Johri
Assistant Professor
Department of Computer Science and Engineering

Submitted By

18SCSE1010128 – PRATYUSH SRIVASTAVA

18SCSE1010030 – ABHISHEK TIWARI

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING GALGOTIAS UNIVERSITY, GREATER NOIDA, INDIA DECEMBER - 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **" Food Calorie Prediction Using Deep Learning "** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the **School of Computing Science and Engineering** of Galgotias University, Greater Noida, is an original work carried out during the period of **JULY-2021 to DECEMBER-2021**, under the supervision of **Dr. Prashant Johri, Assistant Professor**, **Department of Computer Science and Engineering** of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

18SCSE1010128 – PRATYUSH SRIVASTAVA
18SCSE1010030 – ABHISHEK TIWARI

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Supervisor

(Dr.Prashant Johri, Assistant Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **18SCSE1010128 – PRATYUSH SRIVASTAVA, 18SCSE1010030 – ABHISHEK TIWARI** has been held on 18-12-2021_ and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date:

Place:

ABSTRACT

Food is the one of the major need of any human body or We can say that Food is the fuel of human body & one of the basic necessities of human beings. Due to modern life style dietary habits of human being have changed which include consumption of ready mode, packaged & fast food with the reduction of physical labour or exercise carried out by human beings. This kind of unbalanced diet is a high risks factor for diseases & ailments such as obesity, cardiac problems & a host of other diseases. Accurate methods to measure food and energy intake are crucial for the battle against obesity. Providing users/patients with convenient and intelligent solutions that help them measure their food intake and collect dietary information are the most valuable insights toward long- term prevention and successful treatment programs. In this paper, we propose an assistive calorie measurement system to help patients and doctors succeed in their fight against diet-related health conditions.

Our proposed system runs on smartphones, which allow the user to take a picture of the food and measure the amount of calorie intake automatically. In order to identify the food accurately in the system, we use deep convolutional neural networks to classify 10000 high- resolution food images for system training. Our results show that the accuracy of our method for food recognition of single food portions is 99%. Our work is aimed at determination or classification of food using image processing(MobileNet) inconjunction with other intelligent algorithms, with the ultimate aim of determination/estimation of caorie intake our work acts as basis of modern computer assisted, remote dietary management systems. Our system comprises of segmentation of food in the image, then extracting image parameters such as area, major axis, minor axis convex area from the segmented food area, & then using an already trained artificial neural network to classify the food on basis of these parameters. Multiple methods have been combined using weighted averaging to achieve food segmentation, High detection accuracy is obtained by combination of multiple image processing techniques with leven barg marquard function flitting neural network.

Table of Contents

INTRODUCTION

When people's Body Mass Index (BMI) is over thirty (kg/m2), they're usually thought-about to be corpulent. High BMI will increase the chance of diseases like cardiopathy [1]. the most reason of fat is attributable to the imbalance between the number of caloric intake (consumption) and energy output (expenditure). due to disposition to record and track, lack of connected nutritionary info or alternative reasons, patients typically expertise hassle in dominant the number of calories they consume. There square measure countless projected strategies to estimate calories supported laptop vision [2, 3, 4, 5], however once the authors' analysis, the accuracy of detection and volume estimation still ought to be improved. during this paper, the most distinction from alternative similar approaches is that it needs associate input of 2 pictures, and also the use quicker R-CNN to sight the item and GrabCut algorithmic rule to get every food's contour. After that, the authors will estimate every food's volume and calories.

Human services sustenance and great practices indietary patterns pull in individuals' consideration as of late. These days, innovation can enable clients to monitor their sustenance utilization and increment mindfulness in every day diet by observing nourishment propensities. As of late, various research papers have exhibited that machine learning strategies and PC vision procedures can help construct frameworks for programmed acknowledgment of various nourishment and gauge the measure of sustenance [1] - [5]

Accordingly, the purpose of this work is to improve the identification process of fruit and vegetables performed by the self-service systems in the retail market. More specifically, the improvement should consist of a faster process and a more user friendly system. The purpose of implementing computer vision to the system is to narrow the selection of possible objects and thus reduce the strain on the user. Additionally, the use of computer vision in self-service systems can simplify the process of identifying objects by moving the process from a human to a computer. Theoretically, this could hasten the process to identify products and minimize the amount of errors by removing the human factor.

The work and this research paper is to elaborate one algorithm which is not very heavy loaded and provide excellent accuracy as well.For achieving so We

are going to use the Concept of Transfer Learning and MobileNet . Transfer Learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. ... Common examples of transfer learning in deep learning. When to use transfer learning on your own predictive modeling problems. Where as MobileNet-v2 is a convolutional neural network that is 53 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

Motivation:

Computer vision has been introduced to estimate calories from food images. But current food image datasets don't contain volume and mass records of foods, which leads to an incomplete calorie estimation. Current obesity treatment techniques require the patient to record all food intakes per day. In most of the cases, unfortunately patients have troubles in estimating the amount of food intake because of the self-denial of the problem, lack of nutritional information, the manual process of writing down this information (which is tiresome and can be forgotten), and other reasons.

Related work

As of late, it has been exhibited that visual acknowledgment and machine learning strategies can be utilized to create frameworks that keep tracks of human nourishment utilization. The real handiness of these framework intensely relies upon the capacity of perceiving sustenance in unconstrained conditions. In this paper, we proposed another dataset for the assessment of sustenance acknowledgment calculations. The pictures have been obtained in a genuine cafeteria and delineate a genuine cafeteria plate with sustenance masterminded in various ways. Every plate contains various occurrences of nourishment classes. We gathered an arrangement of 1027 plate for an aggregate of 3616 nourishment occurrences having a place with 73 sustenance classes. The plate pictures have been physically sectioned utilizing precisely drawn polygonal limits. We structured a reasonable programmed plate examination pipeline that takes a plate

picture as input, finds the areas of intrigue, and predicts the relating nourishment class for every district. We assessed three distinctive arrangement techniques utilizing a few visual descriptors. The best execution has been gotten by utilizing CNNs-based features. The dataset, and also the benchmark system, are made accessible to the examination network. On account of the manner in which it has been commented on, this database alongside the UNIMIB2015 can be utilized for Sustenance division, acknowledgment, and amount estimation.

The robotic fruit harvesting system is developed with the help of fruit detection algorithm using multiple structures identical intensity, color, alignment and edge of the fruit images. With the help of improved multiple feature based algorithm the detecting effectiveness is attained up to 90% for various fruit items [1]. For the exploration of the image FFB, the expansion of out-of-doors image inspection of oil palm fruit fresh bunches (FFB) are essential. The software examination generates the accurate prototypical and connection component amongst the light intensity in kin to value of FFB from RGB element of image occupied . The on-line valuation of the superiority of fruits the calculation of the effectiveness of these methods concerning the next superiority facets hereby size, color, stem position and recognition of outer flaws is offered . The main stages of the pipeline are segmentation of items from background, feature extraction mainly based on color, and classification with Gaussian Bayes classifier . An automatic spherical fruits recognition system in the natural conditions facing difficult situations such as shadows, bright areas, occlusions and overlapping fruit Convolutional neural Network achieved ample improved than did outdated approaches By means of handcrafted features. Complete comment of competent convolution kernels, we inveterate that color structures are vital to nutrition image identification . Defined nutrition identification consuming a minor dataset, which was proposed to be secondhand in a Smartphone based food classification scheme . Which identifies unhealthy foods beginning cartridges of eating and guess meal calories created on identifying diets Estimating the ideal heaviness to trust diverse image features with MKL, they take attained the 61.34% classification percentage for 50 types of diets through the cross-validation-based assessment

State Of Work

The food classification, previous work was concentrated on basic machine learning algorithms like Random Forest and SVM with hand-tailored features. These methods are generally based on relative or spatial relationships of features. But these methods come with computational cost at a large scale. Random forest came up with an

accuracy of 62% whereas SVM came up with an accuracy of 67% and our CNN model built with TensorFlow gave an accuracy of 97% . Also, we show here how the model behaves with hyper parameter tuning by changing parameters like the learning rate and number of neurons in each hidden layer which govern how effectively a model behaves. The working model if this tensor flow is divided into different layer. That is sub category of L1 and L2 that will be helpful for optimizing the task. Tensorflow's Object detection API to detect multiple food items in each image and then using mathematical calculations to find the calorie content of food classes present in the image There are multiple flows of below approach such as Thumb, Simple Food Images, and Size assessment. So we utilize thumb for the volume expectation utilizing clients, the thumb is anything but a proficient way. Since other than the enlisted client whoever utilizes this than the precision will drop. Just Works on Simple Food Images, this methodology doesn't chip away at complex food varieties like soup, sandwich and so forth it just deals with straightforward food sources like apple, banana and so on . I utilized a dataset of inexpensive food pictures dependent on the Pittsburgh Fast-Food Image Dataset. Mathworks image processing toolbox is utilized for separating highlights . Absolute of 11,868 crude highlights separated from RGB portrayal of the picture. Crude highlights decreased utilizing Principal Component Analysis (PCA) and Information Gain (InfoGain) to 23 highlights . Utilizing these highlights and Sequential Minimal Optimization (SMO) they arranged the food. Size expectation is finished by utilizing Random Forest in grams. At long last, the calorie of the food is anticipated utilizing multilayer perceptron. They investigate various sorts of portrayal of a picture, for example, Averaged RGB, Gray Scale, BW 0.7 and BW 0.5 however they get the best outcomes with RGB portrayal . For the size assessment, the ground truth estimation of the food will be taken from its producer. On account of that this technique gives a low precision on the food sources from different producers. White Background, they utilized a white foundation for the food sources in this tasks dataset. Therefore, this venture isn't effective for day by day. A dataset of 2978 pictures of 19 distinctive food were taken. So I utilized Faster RCNN for object discovery . Each jumping box that made by Faster R-CNN is arranged. Utilizing the GrabCut calculation for picture division I separated food into 3 classes' ellipsoid, segment and unpredictable . Utilizing the coin as a kind of perspective point we can ascertain the volume of the food relying upon its shape. Realizing the volume mass can be handily determined utilizing the thickness of the food. With the mass is realized we can figure the calorie of the food.

DATASETS AND METHODS

A. Datasets

For our paper, annotated images of the Indian meal were required. For this we extracted custom dataset from images.google.com and labelled the images using labelImg. Our custom dataset contains 500 annotated images of 6 classes and split 2/3 and 1/3 in train and validation dataset. 6 classes of the dataset being Bhaji(vegetable), Dal(curry), Rice, Roti, Puri and Gulab Jamun.

A.1 CNN (custom dataset)

We performed binary classification on custom dataset of the Indian Roti which helped us predict a given image to be a Roti or not. In this we used a total of 200 Roti and non-roti images.

A.2 YOLO V2 (COCO dataset)

We used the COCO dataset to train the YOLO model first. It contains 80 object classes such as person, laptop, apple, etc. With 35000 images in the dataset with train, valid, test split in 1/2,1/4,1/4 respectively. Annotations being in XML and text format

A.3 YOLO V3 (custom dataset)

Our custom dataset contains 500 annotated images of 6 classes and split 2/3 and 1/3 in train and validation dataset. 6 classes of the dataset being Sabji (vegetable), Dal (Indian curry), Rice, Roti, Puri and Gulab Jamun.

B. Methods

The first step in our project is to register a plate and a reference object which will be a 1-rupee or 2-rupee or 5-rupee coin and by using OpenCV modules we will calculate the dimensions of the plate that will be further used to give the calorie count of the food item. For the food detection we are using YOLO (You Only Look Once) which is a deep learning algorithm. The dataset will be labelled and trained according to the YOLO format and after the food detection the counted. For all the items except for rice we will be using standard calorie values and for rice we will calculate the area of it and eventually we will get the calorie count through mathematical computations. The formula for the area calculation that we are using is:

Fr = Sr * (Fp /Sp) where,

- Fr: area of target food item.
- Fp: the area of the registered plate.
- Sp: the pixel count of the whole registered plate.
- Sr: the pixel count of the region of the target food item.

After getting the calorie counts of each item the aggregated calorie count of the meal will be presented to the user.

SOFTWARE AND HARDWARE REQUIREMENT

The CNN model is trained on the machine having 8GB (Gigabytes) of RAM and 2GB of VRAM. In this system two programming languages are used namely, Python and Java. The CNN framework consists of Darknet layers. CNN uses many Python libraries wiz. NumPy, OpenCV, Pandas, etc. Whereas for the GPU or VRAM CUDA drivers are used. CNN model is saved as collection of
various entities listing: a Neural Network Model, a custom Configuration file and trained Weights. Java is used for Android application.

Tools and Libraries:

No	Tools & Library Name	Usage
1	Keras	We are using for deep learning tasks like creating model, predicting the object etc.
2	Pillow	Pillow we are using for preprocessing the images of our dataset.
3	Streamit	It is backend framework for developing the web application.
4	Beautifulsoup, Requests	We are using it for scraping the calories from the internet for the predicted object.
5	Numpy	We are using it for the Image matrix handling.

MATERIAL AND STRATEGIES

Deep Learning

Deep learning is a component of a broader conception of machine learning ways supported learning knowledge and its representations. Learning are often carried in 3 ways supervised, semi-supervised or unattended. Deep learning consists of following architectures like deep neural ne process, audio recognition, social network filtering, computational linguistics, bio-informatics, medical image analysis, material scrutiny and parlor game programs, wherever they need made results love and in some cases superior to human specialists.

Deep Learning based mostly Objection Detection

The authors selected quicker R-CNN rather than victimisation linguistics segmentation technique like absolutely Convolution Networks (FCN). Here, once the pictures square measure inputted as RGB channels, the authors will get a series of bounding boxes, which implies the category if judge

LITERATURE SURVEY/PROJECT DESIGN

Paper Name : Machine Learning Based Approach on Food Recognition and Nutrition Estimation

These days, a typical healthy diet it is important to store food to prevent obesity the human body. In this paper, we present in full a unique system supported by machine learning that automatically performs precise food classification photos and measuring food qualities. This paper proposes an in-depth learning model that contains convolutional neural network that separates food specific areas in the training component of the model type system. The main purpose of the proposed approach is to do so improves the accuracy of the pre-training model.

The papers design a model system that supports the client server model. Client sends image detection request and process it on the server side. The prototype system is intended for three major software components, including the training of a pre-trained CNN model module for classification purposes, text data training module for moderation attribute models, as well as server module. We tried food distribution categories, each containing thousands of images, too through machine learning training for maximum achievement section accuracy.

Paper Name : Deep Food: Food Image Analysis and Dietary Assessment via Deep Model

Food is important to human health and has been a priority in many health meetings.These days new food testing and food analysis tools offer many opportunities to help people understand their daily eating habits, and check nutrition patterns and keep a healthy diet. In this paper, we develop an in-depth model of food and food recognition a food review and analysis program from

everyday food photos (e.g., taken by a smartphone).Specifically, we propose a three-step algorithm to accept photos of many (food) items by discovery regions to be immersed and use a deep convolutional neural network (CNN) object division. The program first creates a wide range of suggestions for input images using the Regional Proposal Network (RPN) obtained from the Faster R-CNN model. It then identifies each region of suggestions by marking them on feature maps, and they divide themselves into different categories of food, and as placing them within the original images. Finally,The program will analyze the ingredients for healthy eating supports the effects of popularity and creates food calorie counting report,fats, carbohydrates and proteins. In testing, we do extensive exercises using two popular foods Image data sets - UEC-FOOD100 and UEC-FOOD256 and generate a new type of data about food items that are supported by FOOD101 by binding. The model is tested with different test metrics. The test results show that our system is in the position of receiving food items accurately and produce a good food test report, viz will bring users out with a clear view of life diet and guide their daily bodybuilding recipe health and well-being.

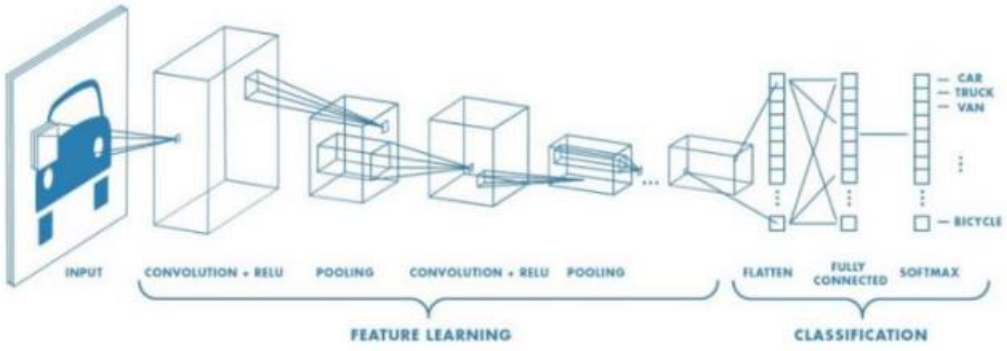
PROPOSED SYSTEM

Food recognition is an existing idea which can detect and recognize food item based on the input image. Our model is trained on 101 categories of food items. Further the idea is to estimate the calorie of the food item which is being recognized. The convolutional Neural Network (CNN) is used to recognize the food item. Further to estimate the calories we have given the standard calorie value for one gram of each food item. The weight of the food item is given as an input and based on the standard calorie value the accurate calorie value of the food item is calculated.

Recognition method

Food Recognition deals with recognition of food item when given an image.For this problem I used Convolutional Neural Network (CNN). The Architecture of CNN given below figure 2

Figure 2: Architecture of CNN



For this project I used 5 convolution layers with ReLU activations,dropout, and softmax layers. Fine tuning the model on our dataset took about 2 hours on a single Windows 10 Pro CPU with 4GB of memory.For this training I used 100 imagesof each food with 300*300 image size. All this work done in Python 3.7.1 with Anaconda Distribution 4.6.11

Also used Adam optimizer and categorical cross entropy loss function with learning rate 0.0001 to calculate and minimize loss as well as optimize model accuracy

METHODOLOGY

The project consists of two steps, identifying food from an image and converting the food identified into a calorie estimation. We performed food image classification using CNN (convolutional Neural Network). Steps followed:

1. **Pre-processing**: Some basic pre-processing has been performed to clean the dataset where the irrelevant and noisy images of 15 categories have been removed.

Also, data augmentation has been performed –

- Pixel values re-scaled in the range of [0,1].
- Random rotations max 40 degree.
- Random zoom applied.

- Shear angle in counter-clockwise direction in degrees

2**. Trained the model:** We trained the model with images of 15 categories using the classifier CNN
(convolutional Neural Network) which is a class of deep, feed forward artificial neural networks that has
successfully been applied to analyzing visual imagery

Convolutional Neural Network

The convolutional Neural Network (CNN) offers a technique for many general image classification problems. It has been applied in food classification and resulted in a good accuracy.CNN is widely used in food recognition and provides high performance than the traditional methods. Over the last few years, due to the enhancements in the deep learning, especially in the convolutional neural networks, the accuracy in detecting and recognizing food images has been increased. This is not only because larger datasets but also new algorithms and improved deep architectures. convolutional Neural Network (CNN) is also known as LeNet due to its inventor.CNN mainly comprises convolutional layers, pooling layers and sub-

sampling layers followed by fully-connected layers. The CNN takes an input image and applies convolutional and then sub-sampling. After two such computations, the data is fed into the fully connected neural network, where it performs the classification task. The main advantage of CNN is the ability to learn the high-level efficient features and in addition to that, it is robust against small rotations and shifts.

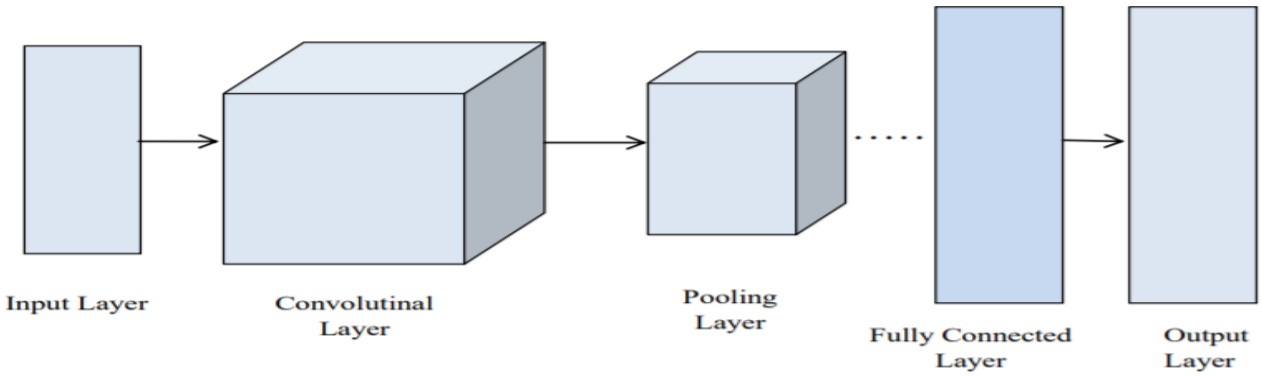


Image classification!

The convolutional neural network (CNN) is a class of **deep learning neural networks**. CNNs represent a huge breakthrough in image recognition. They're most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook's photo tagging to self-driving cars. They're working hard behind the scenes in everything from healthcare to security.

They're fast and they're efficient. But how do they work?

Image classification is the process of taking an **input** (like a picture) and outputting a **class** (like "cat") or a **probability** that the input is a particular class ("there's a 90% probability that this input is a cat"). You can look at a picture and know that you're looking at a terrible shot of your own face, but how can a computer learn to do that?

With a convolutional neural network!

A CNN has

- Convolutional layers
- ReLU layers

- Pooling layers
- a Fully connected layer

A classic CNN architecture would look something like this:

Input ->Convolution ->ReLU ->Convolution ->ReLU ->Pooling ->
ReLU ->Convolution ->ReLU ->Pooling ->Fully Connected

A CNN **convolves** (not convolutes…) learned features with input data and uses 2D convolutional layers. This means that this type of network is ideal for processing 2D images. Compared to other image classification algorithms, CNNs actually use very little preprocessing. This means that they can **learn** the filters that have to be hand-made in other algorithms. CNNs can be used in tons of applications from image and video recognition, image classification, and recommender systems to natural language processing and medical image analysis.

CNNs are inspired by biological processes. They're based on some cool research done by Hubel and Wiesel in the 60s regarding vision in cats and monkeys. The pattern of connectivity in a CNN comes from their research regarding the organization of the visual cortex. In a mammal's eye, individual neurons respond to visual stimuli only in the receptive field, which is a restricted region. The receptive fields of different regions partially overlap so that the entire field of vision is covered. This is the way that a CNN works.

CNNs have an input layer, and output layer, and hidden layers. The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.

- Convolutional layers apply a convolution operation to the input. This passes the information on to the next layer.
- Pooling combines the outputs of clusters of neurons into a single neuron in the next layer.
- Fully connected layers connect every neuron in one layer to every neuron in the next layer.

In a convolutional layer, neurons only receive input from a subarea of the previous layer. In a fully connected layer, each neuron receives input from *every* element of the previous layer.

A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained! They're learned while the network trains on a set of images. This makes deep learning models extremely accurate for computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

A CNN

- starts with an input image
- applies many different filters to it to create a feature map
- applies a ReLU function to increase non-linearity
- applies a pooling layer to each feature map
- flattens the pooled images into one long vector.
- inputs the vector into a fully connected artificial neural network.
- processes the features through the network. The final fully connected layer provides the "voting" of the classes that we're after.
- trains through forward propagation and backpropagation for many, many epochs. This repeats until we have a well-defined neural network with trained weights and feature detectors.

So what does that mean?

At the very beginning of this process, an input image is broken down into pixels. Based on that information, the computer can begin to work on the data.

For a color image, this is a 3D array with a blue layer, a green layer, and a red layer. Each one of those colors has its own value between 0 and 255. The color can be found by combining the values in each of the three layers.

What are the basic building blocks of a CNN?

Convolution

The main purpose of the convolution step is to extract features from the input image. The convolutional layer is always the first step in a CNN.

You have an input image, a feature detector, and a feature map. You take the filter and apply it pixel block by pixel block to the input image. You do this through the multiplication of the matrices.

The light from the flashlight here is your **filter** and the region you're sliding over is the **receptive field**. The light sliding across the receptive fields is your flashlight **convolving**. Your filter is an array of numbers (also called weights or parameters). The distance the light from your flashlight slides as it travels (are you moving your filter over one row of bubbles at a time? Two?) is called the **stride**. For example, a stride of one means that you're moving your filter over one pixel at a time. The convention is a stride of two.

The depth of the filter has to be the same as the depth of the input, so if we were looking at a color image, the depth would be 3. That makes the dimensions of this filter 5x5x3. In each position, the filter multiplies the values in the filter with the original values in the pixel. This is element wise multiplication. The multiplications are summed up, creating a single number. If you started at the top left corner of your

bubble wrap, this number is representative of the top left corner. Now you move your filter to the next position and repeat the process all around the bubble wrap. The array you end up with is called a **feature map** or an **activation map**! You can use more than one filter, which will do a better job of preserving spatial relationships.

Transfer Learning:

Transfer learning means taking the relevant parts of a pre-trained machine learning model and applying it to a new but similar problem. This will usually be the core information for the model to function, with new aspects added to the model to solve a specific task.

Transfer Learning for Image Recognition

A range of high-performing models have been developed for image classification and demonstrated on the annual ImageNet Large Scale Visual Recognition Challenge, or ILSVRC.

This challenge, often referred to simply as ImageNet, given the source of the image used in the competition, has resulted in a number of innovations in the architecture and training of convolutional neural networks. In addition, many of the models used in the competitions have been released under a permissive license.

These models can be used as the basis for transfer learning in computer vision applications.

This is desirable for a number of reasons, not least:

- **Useful Learned Features**: The models have learned how to detect generic features from photographs, given that they were trained on more than 1,000,000 images for 1,000 categories.
- **State-of-the-Art Performance**: The models achieved state of the art performance and remain effective on the specific image recognition task for which they were developed.
- **Easily Accessible**: The model weights are provided as free downloadable files and many libraries provide convenient APIs to download and use the models directly.

The model weights can be downloaded and used in the same model architecture using a range of different deep learning libraries, including Keras.

How to Use Pre-Trained Models

The use of a pre-trained model is limited only by your creativity.

For example, a model may be downloaded and used as-is, such as embedded into an application and used to classify new photographs.

Alternately, models may be downloaded and use as feature extraction models. Here, the output of the model from a layer prior to the output layer of the model is used as input to a new classifier model.

Recall that convolutional layers closer to the input layer of the model learn low-level features such as lines, that layers in the middle of the layer learn complex abstract features that combine the lower level features extracted from the input, and layers closer to the output

interpret the extracted features in the context of a classification task.

Armed with this understanding, a level of detail for feature extraction from an existing pre-trained model can be chosen. For example, if a new task is quite different from classifying objects in photographs (e.g. different to ImageNet), then perhaps the output of the pre-trained model after the few layers would be appropriate. If a new task is quite similar to the task of classifying objects in photographs, then perhaps the output from layers much deeper in the model can be used, or even the output of the fully connected layer prior to the output layer can be used.

The pre-trained model can be used as a separate feature extraction program, in which case input can be pre-processed by the model or portion of the model to a given an output (e.g. vector of numbers) for each input image, that can then use as input when training a new model.

Alternately, the pre-trained model or desired portion of the model can be integrated directly into a new neural network model. In this usage, the weights of the pre-trained

can be frozen so that they are not updated as the new model is trained. Alternately, the weights may be updated during the training of the new model, perhaps with a lower learning rate, allowing the pre-trained model to act like a weight initialization scheme when training the new model.

We can summarize some of these usage patterns as follows:

- **Classifier**: The pre-trained model is used directly to classify new images.
- **Standalone Feature Extractor**: The pre-trained model, or some portion of the model, is used to pre-process images and extract relevant features.
- **Integrated Feature Extractor**: The pre-trained model, or some portion of the model, is integrated into a new model, but layers of the pre-trained model are frozen during training.
- **Weight Initialization**: The pre-trained model, or some portion of the model, is integrated into a new model, and the layers of the pre-trained model are trained in concert with the new model.

Each approach can be effective and save significant time in developing and training a deep convolutional neural network model.

It may not be clear as to which usage of the pre-trained model may yield the best results on your new computer vision task, therefore some experimentation may be required.

Models for Transfer Learning

There are perhaps a dozen or more top-performing models for image recognition that can be downloaded and used as the basis for image recognition and related computer vision tasks.

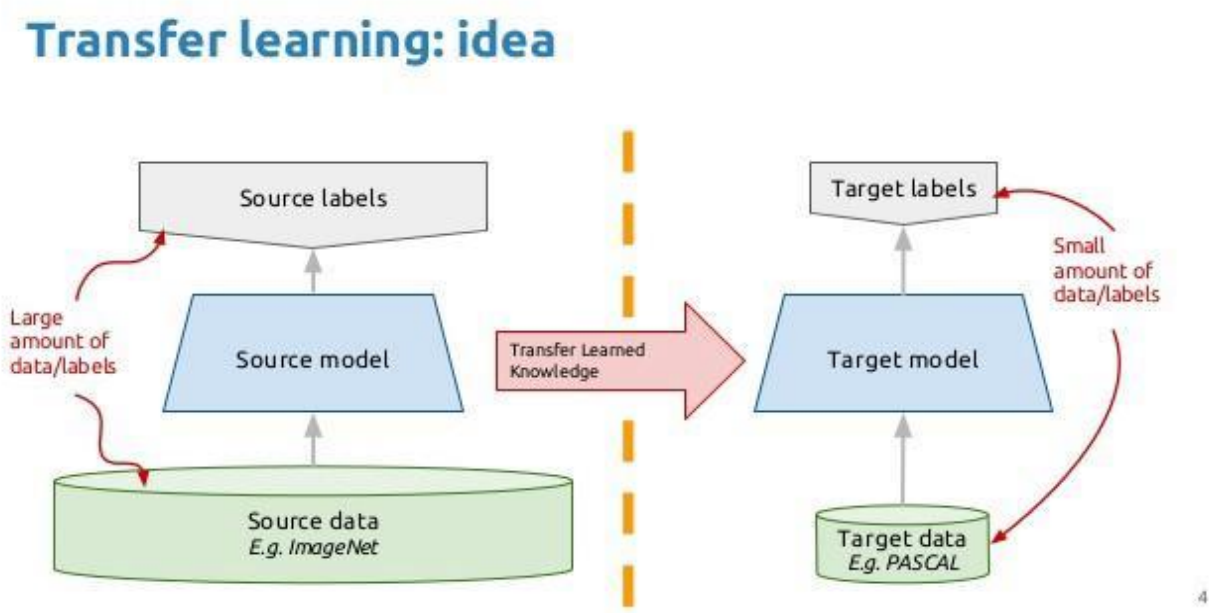
Perhaps three of the more popular models are as follows:

- VGG (e.g. VGG16 or VGG19).
- GoogLeNet (e.g. InceptionV3).
- Residual Network (e.g. ResNet50).

These models are both widely used for transfer learning both because of their performance, but also because they were examples that introduced specific architectural innovations, namely consistent and repeating structures (VGG), inception modules (GoogLeNet), and residual modules (ResNet).

Keras provides access to a number of top-performing pre-trained models that were developed for image recognition tasks.

They are available via the Applications API, and include functions to load a model with or without the pre-trained weights, and prepare data in a way that a given model may expect (e.g. scaling of size and pixel values).
The first time a pre-trained model is loaded, Keras will download the required model weights, which may take some time given the speed of your internet connection.
Weights are stored in the *.keras/models/* directory under your home directory and will be loaded from this location the next time that they are used.
When loading a given model, the "*include_top*" argument can be set to *False*, in which case the fully-connected output layers of the model used to make predictions is not loaded, allowing a new output layer to be added and trained.



Following is the general outline for transfer learning for object recognition:

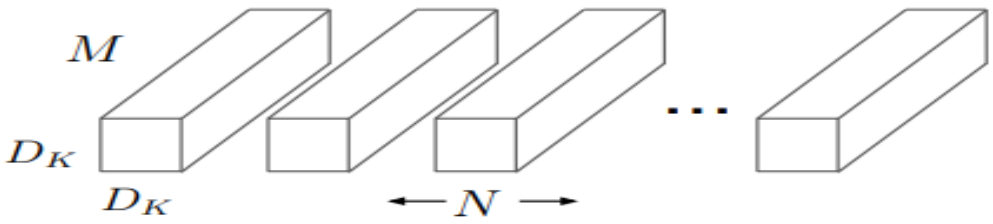
- Load in a pre-trained CNN model trained on a large dataset
- Freeze parameters (weights) in model's lower convolutional layers
- Add custom classifier with several layers of trainable parameters to model
- Train classifier layers on training data available for task
- Fine-tune hyperparameters and unfreeze more layers as needed

This approach has proven successful for a wide range of domains. It's a great tool to have in your arsenal and generally the first approach that should be tried when confronted with a new image recognition problem.

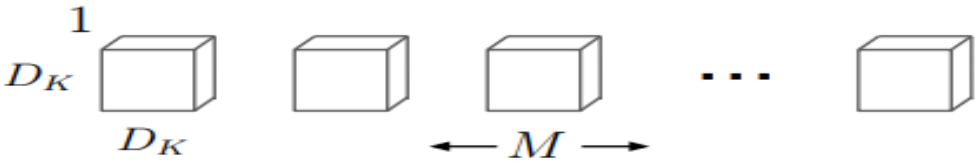
MobileNet:

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Howard et al, 2017.

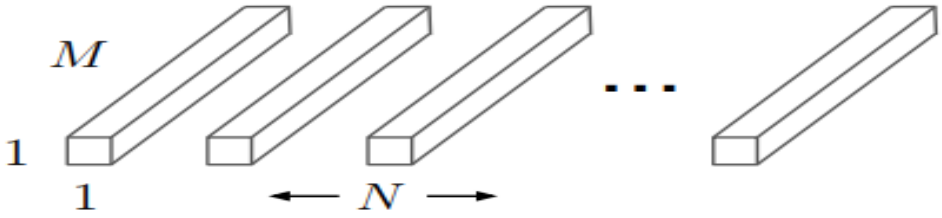
We shall be using Mobilenet as it is lightweight in its architecture. It uses depthwise separable convolutions which basically means it performs a single convolution on each colour channel rather than combining all three and flattening it. This has the effect of filtering the input channels. Or as the authors of the paper explain clearly: " For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs the depthwise convolution. A standard convolution both filters and combines inputs into a new set of outputs in one step. The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size. "



(a) Standard Convolution Filters



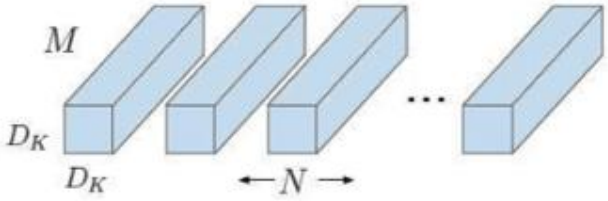
(b) Depthwise Convolutional Filters



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

MobileNet Architecture:

The main aim of TL is to implement a model quickly. There will be no change in the MobileNet architecture whatsoever. The model will transfer the features it has learned from a different dataset that has performed the same task.



Standard Convolution Filters

Computational cost = $D_f^2 * M * N * D_K^2$

Where:
D_f^2- Dimension of input feature maps
M- Input Channel
N- Output Channel
D_K^2- Kernel size

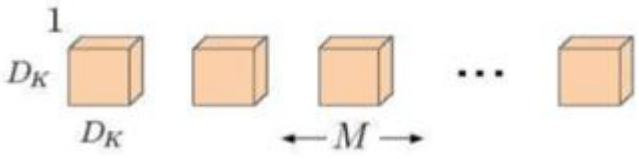
While applying the composite function in the standard convolution layer, the convolution kernel is applied to all the channels of the input image and slides the weighted sum to the next pixel. MobileNet uses this standard convolution filter on only the first layer.

Depthwise Separable Convolution

The next layer is *depthwise separable convolution*, which is the combination of *depthwise* and *pointwise* convolution.

Depthwise Convolution

Unlike standard convolution, a depthwise convolution maps only one convolution on each input channel separately. The channel dimension of the output image (3 RGB) will be the same as that of an input image.

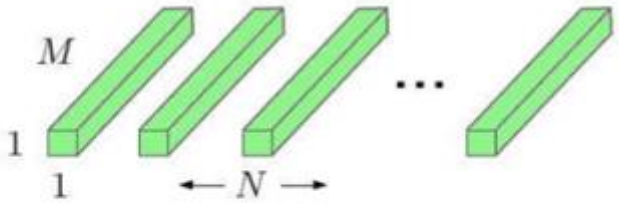


Depthwise Convolution Filters

Computational cost = $D_f^2 * M * D_K^2$

Pointwise Convolution

This is the last operation of the filtering stage. It's more or less similar to regular convolution but with a 1x1 filter. The idea behind pointwise convolution is to merge the features created by depthwise convolution, which creates new features.

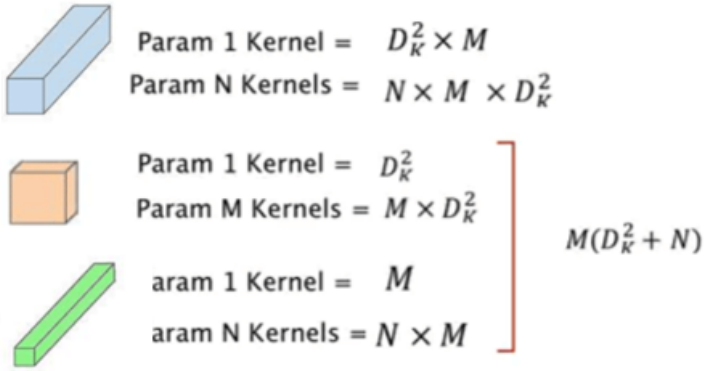


Pointwise Convolution Filters (1x1 conv)

Computational cost = $D_f^2 * M * N$

The cost function of DSC is the sum of the cost of depthwise and pointwise convolution.

Depthwise Separable Convolution Cost Function



$$\frac{Number\ of\ params\ in\ DSC}{Number\ of\ param\ in\ standard\ Conv} = \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

times less operations than standard convolution

Ref: Howard et al

Other than this, MobileNet offers two more parameters to reduce the operations further:

Width Multiplier:

This introduces the variable α ∈ (0, 1) to thin the number of channels. Instead of producing N channels, it will produce αxN channels. It will choose 1 if you need a smaller model.Resolution Multiplier: This introduces the variable ρ ∈ (0,

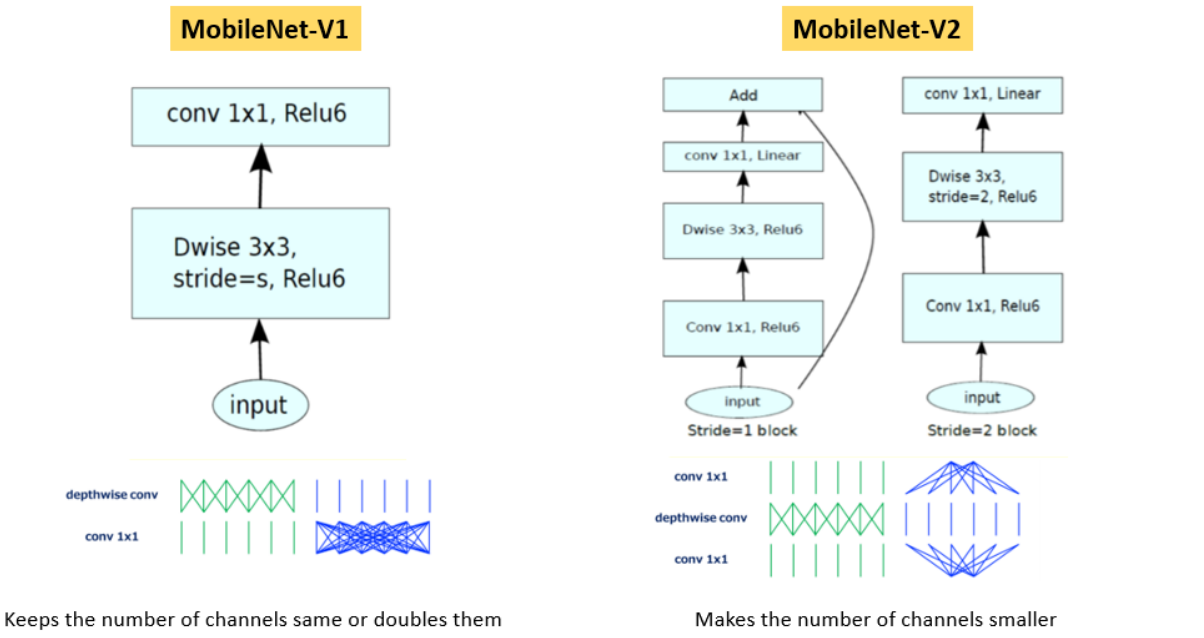
1), it is used to reduce the size of the input image from 244, 192, 160px or 128px. 1 is the baseline for image size 224px.

You can train the model on a 224x224 image and then use it on 128x128 images as MobileNet uses Global Average Pooling and doesn't flatten layers.

MobileNet-V2

The MobileNet-V2 pre-trained version is available here. Its weights were initially obtained by training on the ILSVRC-2012-CLS dataset for image classification (Imagenet).

The basic building blocks in MobileNet-V1 and V2:



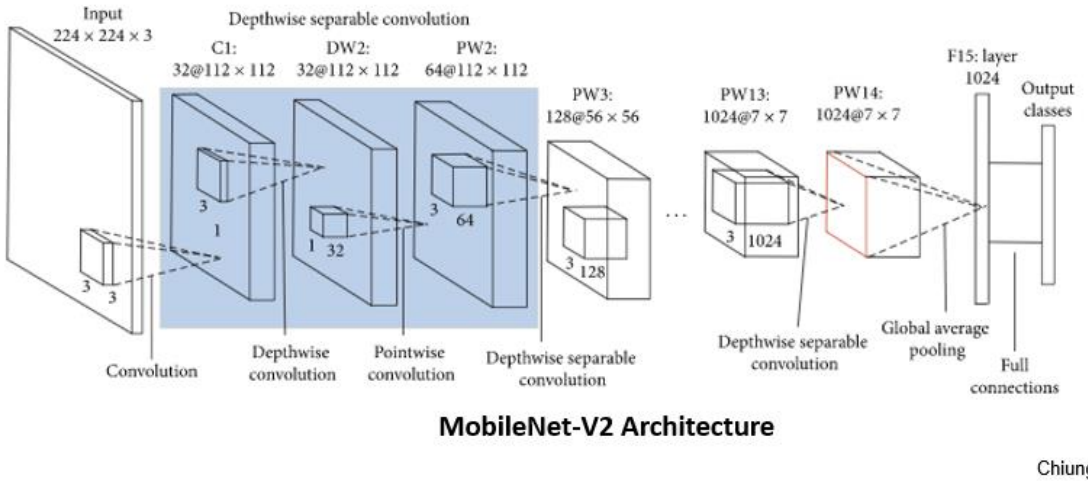
Keeps the number of channels same or doubles them

Makes the number of channels smaller

Howard et al
Yusuke Uchida

MobileNet version

The final MobileNet-V2architecture looks like this:



Difference between pointwise and depth wise convolutions

So the overall architecture of the Mobilenet is as follows, having 30 layers with

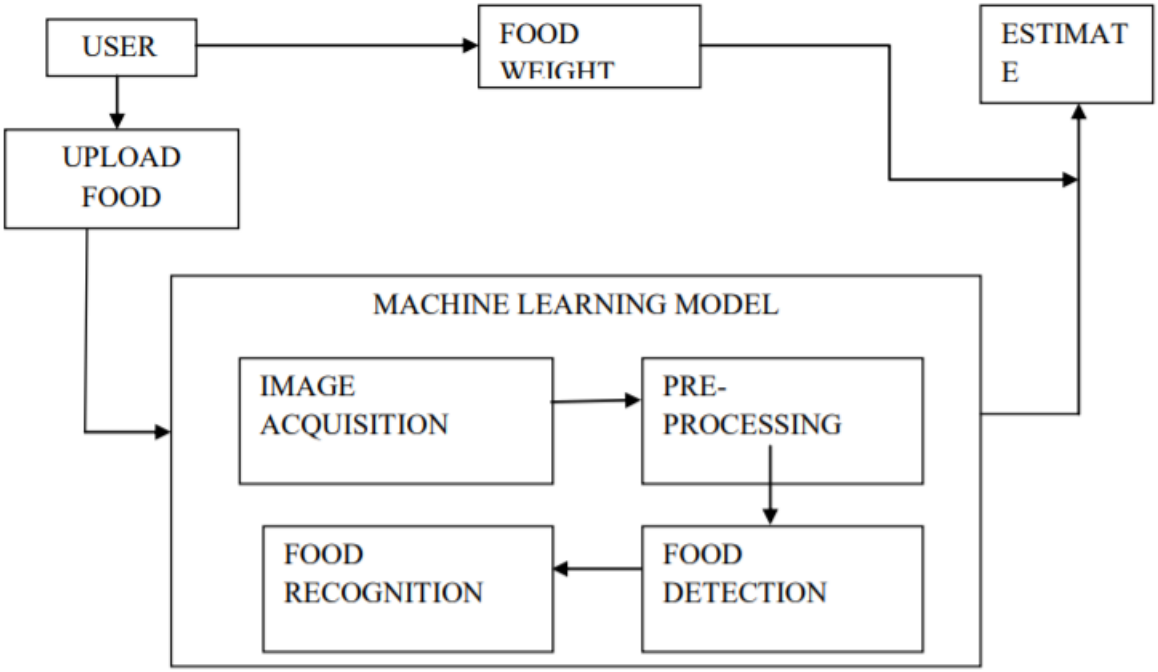
- convolutional layer with stride 2 depthwise layer
- pointwise layer that doubles the number of channels
- depthwise layer with stride 2
- pointwise layer that doubles the number of channels etc.

Table 1. MobileNet Body Architecture

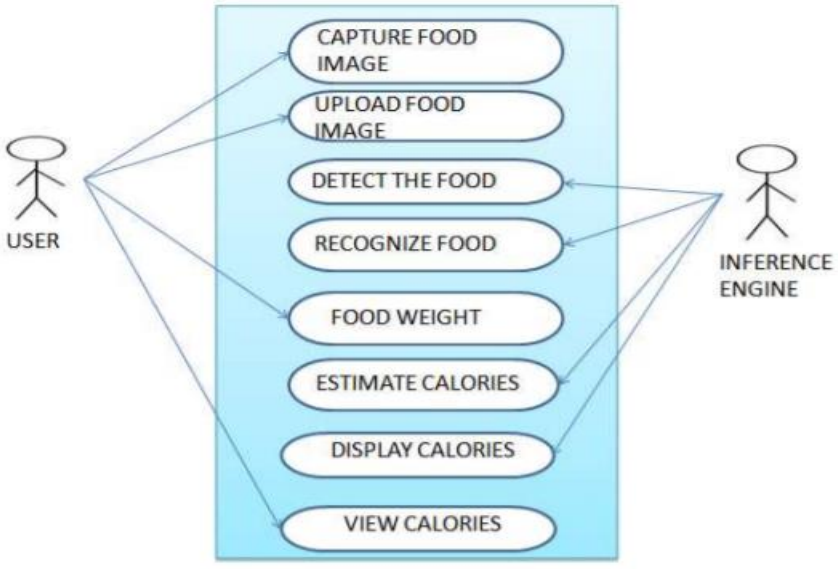
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times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

It is also very low maintenance thus performing quite well with high speed. There are also many flavours of pre-trained models with the size of the network in memory and on disk being proportional to the number of parameters being used. The speed and power consumption of the network is proportional to the number of MACs (Multiply-Accumulates) which is a measure of the number of fused Multiplication and Addition operations.

Architecture Diagram



Use-Case Diagram



Use-Case Diagram

- **Architecture:** - We are using the MobilenetV2 architecture. MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. Mobilenet support any input size greater than 32 x 32
- In MobileNetV2, there are two types of blocks. One is residual block with stride of another one is block with stride of 2 for downsizing.
- There are 3 layers for both types of blocks.
- This time, the first layer is 1×1 convolution with ReLU6.
- The second layer is the depth wise convolution.
- The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if RELU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.

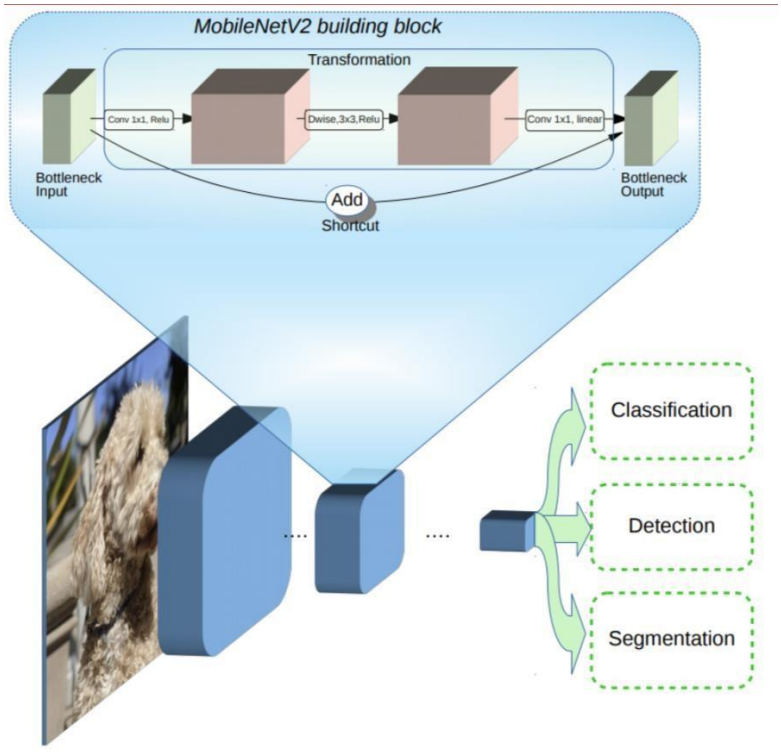


Figure 1. MobilenetV2 Architecture

- Dataset: - In this project we are using the "Fruit and Vegetable Image Recognition" dataset. This dataset have 36 classes, and almost 100 images for each class so we can say we have 3600+ training images. We have 10 images for each category in Train/Validation

- Workflow: - In this we are going to see how our web-application is working. We have divided our modules so our task is going to be easy. Our frontend-backend will be handled by the Streamlit. As a normal user, user will visit our application by URL. There will be upload button so user can upload the image. After the uploading the Image our system will do the task automatically.
- User, will upload the Image. That image will be stored into the local system.
- Now pillow will resize the image according to our model shape, it will convert into vector.
- Now this vector will be passed to our model, our model will classify the class of category.
- We will get the ID of category, now we need to map the labels according to the ID.
- Now our system will do web-scrap the calories for predicted object. Our application will display the Result and Calories into our application.

Source Code and Outputs:

import the necessary libraries. (Remember the TensorFlow version, because we need to use the same version into the local machine)

```
import numpy as np
import pandas as pd
from pathlib import Path
import os.path
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img,img_to_array
print(tf.__version__)
```

```
2.6.0
```

Now we need to define the Train, test, and validation images.

```
# Create a list with the filepaths for training and testing
train_dir = Path('../input/fruit-and-vegetable-image-recognition/train')
train_filepaths = list(train_dir.glob(r'**/*.jpg'))
```

```
test_dir = Path('../input/fruit-and-vegetable-image-recognition/test')
test_filepaths = list(test_dir.glob(r'**/*.jpg'))
```

```
val_dir = Path('../input/fruit-and-vegetable-image-recognition/validation')
val_filepaths = list(test_dir.glob(r'**/*.jpg'))
```

Now we need to create the data frame for each image with its label, So this is the function for it.

```
def image_processing(filepath):
    """ Create a DataFrame with the filepath and the labels of the pictures
    """

    labels = [str(filepath[i]).split("/")[-2] \
              for i in range(len(filepath))]

    filepath = pd.Series(filepath, name='Filepath').astype(str)
    labels = pd.Series(labels, name='Label')

    # Concatenate filepaths and labels
    df = pd.concat([filepath, labels], axis=1)

    # Shuffle the DataFrame and reset index
    df = df.sample(frac=1).reset_index(drop = True)

    return df
```

Now we need to make a function call for Train, Test, and Val images. Let's check how many labels and images we have in the dataset.

```
train_df = image_processing(train_filepaths)
test_df = image_processing(test_filepaths)
val_df = image_processing(val_filepaths)
```

```
print('-- Training set --\n')
print(f'Number of pictures: {train_df.shape[0]}\n')
print(f'Number of different labels: {len(train_df.Label.unique())}\n')
print(f'Labels: {train_df.Label.unique()}')
```

```
-- Training set --

Number of pictures: 3193

Number of different labels: 36

Labels: ['grapes' 'pomegranate' 'banana' 'carrot' 'garlic' 'onion' 'pineapple'
 'orange' 'capsicum' 'jalepeno' 'paprika' 'watermelon' 'raddish' 'lettuce'
 'spinach' 'tomato' 'sweetpotato' 'cauliflower' 'bell pepper' 'peas'
 'lemon' 'kiwi' 'chilli pepper' 'cabbage' 'turnip' 'eggplant' 'potato'
 'soy beans' 'pear' 'mango' 'beetroot' 'sweetcorn' 'cucumber' 'corn'
 'apple' 'ginger']
```

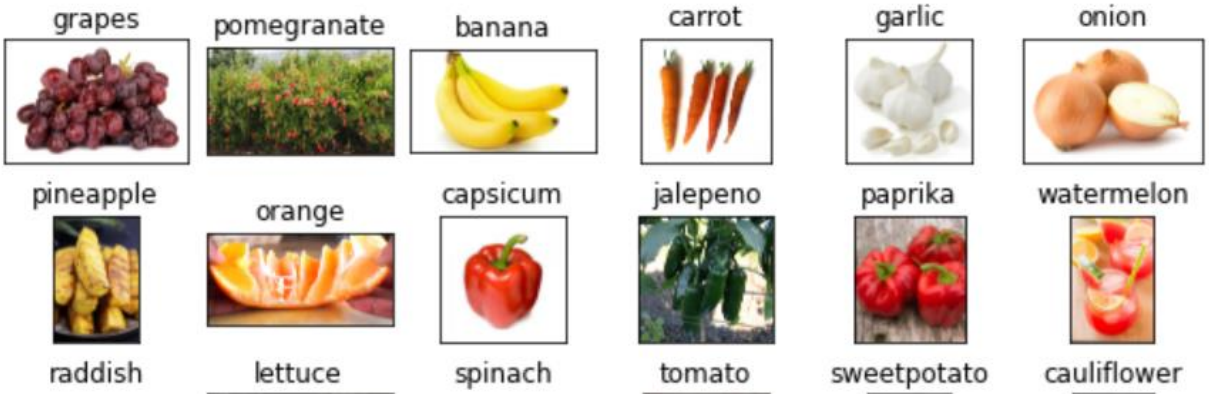
Now let's check the generated data frame.

```
train_df.head(5)
```

	Filepath	Label
0	../input/fruit-and-vegetable-image-recognition...	grapes
1	../input/fruit-and-vegetable-image-recognition...	pomegranate
2	../input/fruit-and-vegetable-image-recognition...	banana
3	../input/fruit-and-vegetable-image-recognition...	carrot
4	../input/fruit-and-vegetable-image-recognition...	garlic

Now let's check our labels of images are matching with the original image or not, We are going to use matplotlib to plot the images.

```
# Create a DataFrame with one Label of each category
df_unique = train_df.copy().drop_duplicates(subset=["Label"]).reset_index()

# Display some pictures of the dataset
fig, axes = plt.subplots(nrows=6, ncols=6, figsize=(8, 7),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(df_unique.Filepath[i]))
    ax.set_title(df_unique.Label[i], fontsize = 12)
plt.tight_layout(pad=0.5)
plt.show()
```



Now we need to generate the new images using these images, because we have a low number of images for each class, we are going to use the

ImageDataGenerator module from the Keras, basically it will do the zoom, rotate, changing the color format, changing brightness and much more technique. With these techniques, it will generate new images.

```
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input
)
```

Let's fit the images for training and testing.

```
train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=0,
    rotation_range=30,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest"
)
```

```
Found 3193 validated image filenames belonging to 36 classes.
```

```
test_images = test_generator.flow_from_dataframe(
    dataframe=test_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=False
)
```

```
Found 334 validated image filenames belonging to 36 classes.
```

Let's fit the pre-trained MobilenetV2 model.

```
pretrained_model = tf.keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
pretrained_model.trainable = False
```

```
User settings:

   KMP_AFFINITY=granularity=fine,verbose,compact,1,0
   KMP_BLOCKTIME=0
   KMP_SETTINGS=1
   KMP_WARNINGS=0
```

Let's start training with our own images.

```
inputs = pretrained_model.input

x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
x = tf.keras.layers.Dense(128, activation='relu')(x)

outputs = tf.keras.layers.Dense(36, activation='softmax')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

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

history = model.fit(
    train_images,
    validation_data=val_images,
    batch_size = 32,
    epochs=5,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=2,
            restore_best_weights=True
        )
    ]
)
```

```
Epoch 1/5
  2/100 [..............................] - ETA: 2:33 - loss: 3.6501 - accuracy: 0.0469
/opt/conda/lib/python3.7/site-packages/PIL/Image.py:963: UserWarning: Palette images with Transparency expressed in bytes should be c
ages
  "Palette images with Transparency expressed in bytes should be "
100/100 [==============================] - 226s 2s/step - loss: 1.9120 - accuracy: 0.4933 - val_loss: 0.4976 - val_accuracy: 0.8383
Epoch 2/5
100/100 [==============================] - 189s 2s/step - loss: 0.7049 - accuracy: 0.7820 - val_loss: 0.2377 - val_accuracy: 0.9341
Epoch 3/5
100/100 [==============================] - 175s 2s/step - loss: 0.4369 - accuracy: 0.8675 - val_loss: 0.1612 - val_accuracy: 0.9521
Epoch 4/5
100/100 [==============================] - 182s 2s/step - loss: 0.2701 - accuracy: 0.9226 - val_loss: 0.1267 - val_accuracy: 0.9551
Epoch 5/5
100/100 [==============================] - 184s 2s/step - loss: 0.1790 - accuracy: 0.9417 - val_loss: 0.1094 - val_accuracy: 0.9521
```

Let's check our trained model on the test images.

```
# Predict the label of the test_images
pred = model.predict(test_images)
pred = np.argmax(pred,axis=1)
# Map the label
labels = (train_images.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred1 = [labels[k] for k in pred]
pred1
```

```
'bell pepper',
'carrot',
'cauliflower',
'paprika',
'mango',
'jalepeno',
'pear',
'beetroot',
'raddish',
'sweetpotato',
```

Now we need to feed some random images from google or from the validation images. Let's create a separate function for it,

```
def output(location):
    img=load_img(location,target_size=(224,224,3))
    img=img_to_array(img)
    img=img/255
    img=np.expand_dims(img,[0])
    answer=model.predict(img)
    y_class = answer.argmax(axis=-1)
    y = " ".join(str(x) for x in y_class)
    y = int(y)
    res = labels[y]
    return res
```

```
img = output('../input/fruit-and-vegetable-image-recognition/test/cabbage/Image_1.jpg')
img
```

```
'cabbage'
```

Let's save the model so we can use it in our application.

```
model.save('FV.h5')
```

Now we need to create a web app using Streamlit.

Let's create an application code. It contains the following things.

- GUI for app

- Model processing code

- Image saving and prediction

- Fetch the calories from google for a particular class.

```
imp
ort
strea
mlit
as st
        from PIL import Image
        from keras.preprocessing.image import load_img,img_to_array
```

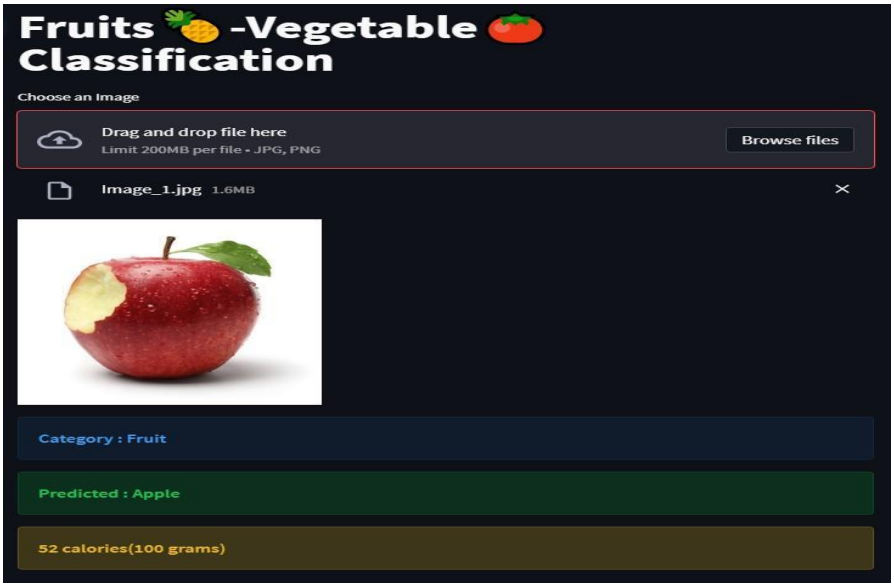
```
import numpy as np
from keras.models import load_model
import requests
from bs4 import BeautifulSoup


model = load_model('FV.h5')
labels = {0: 'apple', 1: 'banana', 2: 'beetroot', 3: 'bell pepper', 4: 'cabbage', 5:
'capsicum', 6: 'carrot', 7: 'cauliflower', 8: 'chilli pepper', 9: 'corn', 10:
'cucumber', 11: 'eggplant', 12: 'garlic', 13: 'ginger', 14: 'grapes', 15: 'jalepeno',
16: 'kiwi', 17: 'lemon', 18: 'lettuce',
        19: 'mango', 20: 'onion', 21: 'orange', 22: 'paprika', 23: 'pear', 24: 'peas',
25: 'pineapple', 26: 'pomegranate', 27: 'potato', 28: 'raddish', 29: 'soy beans',
30: 'spinach', 31: 'sweetcorn', 32: 'sweetpotato', 33: 'tomato', 34: 'turnip', 35:
'watermelon'}


fruits = ['Apple','Banana','Bello Pepper','Chilli
Pepper','Grapes','Jalepeno','Kiwi','Lemon','Mango','Orange','Paprika','Pear','P
ineapple','Pomegranate','Watermelon']
vegetables =
['Beetroot','Cabbage','Capsicum','Carrot','Cauliflower','Corn','Cucumber','Egg
plant','Ginger','Lettuce','Onion','Peas','Potato','Raddish','Soy
Beans','Spinach','Sweetcorn','Sweetpotato','Tomato','Turnip']


def fetch_calories(prediction):
    try:

        url = 'https://www.google.com/search?&q=calories in ' + prediction
        req = requests.get(url).text
        scrap = BeautifulSoup(req, 'html.parser')
        calories = scrap.find("div", class_="BNeawe iBp4i AP7Wnd").text
        return calories
    except Exception as e:
        st.error("Can't able to fetch the Calories")
        print(e)


def processed_img(img_path):
    img=load_img(img_path,target_size=(224,224,3))
```

```
    img=img_to_array(img)
    img=img/255
    img=np.expand_dims(img,[0])
    answer=model.predict(img)
    y_class = answer.argmax(axis=-1)
    print(y_class)
    y = " ".join(str(x) for x in y_class)
    y = int(y)
    res = labels[y]
    print(res)
    return res.capitalize()


def run():
    st.title("Fruits-Vegetable Classification")
    img_file = st.file_uploader("Choose an Image", type=["jpg", "png"])
    if img_file is not None:
        img = Image.open(img_file).resize((250,250))
        st.image(img,use_column_width=False)
        save_image_path = './upload_images/'+img_file.name
        with open(save_image_path, "wb") as f:
            f.write(img_file.getbuffer())


        # if st.button("Predict"):
        if img_file is not None:
            result= processed_img(save_image_path)
            print(result)
            if result in vegetables:
                st.info('**Category : Vegetables**')
            else:
                st.info('**Category : Fruit**')
            st.success("**Predicted : "+result+'**')
            cal = fetch_calories(result)
            if cal:
                st.warning('**'+cal+'(100 grams)**')


run()
```

Now type the following command to run the code in CMD.

streamlit run Streamlit_App.py
Go to the localhost, and upload the image into the app.

Display Output



Results:

In this process 1st we run the main code then the GUI will display. In this we see input image and other options. Then we selelct the input image.Then the output will be displayed.

After this their is two more predictions through mobileNet one of them is the category of the food item and second is type of food and third is most important result is the calorie of that food item.

Image Parameters:

In this process 1st we take the input image of food. After selection of input image a command window will be open. This window shows the parameters of food. Then we follow this process for different-different food image and the resultant parameters are given in the table 1 which is given below.

Table 1. MobileNet Body Architecture

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

Displayed Output:

The displayed is based on Three categories:
→ Taking image as an input
→ Prediction of Image Category
–>Prediction of Type of Image Category
→Prediction of Calorie of that Food Image

Create Web-app file into local machine. Use a saved model to recognize the image.
In this we are going to see how our web-application is working. We have divided our modules so our task is going to be easy. Our frontend-backend will be handled by the Streamlit. As a normal user, user will visit our application by URL. There will be upload button so user can upload the image. After the uploading the Image our system will do the task automatically. User, will upload the Image. That image will be stored into the local system. Now pillow will resize the image according to our model shape, it will convert into vector. Now this vector will be passed to our model, our model will classify the class of category. We will get the ID of category, now we need to map the labels according to the ID. Now our system will do web-scrap the calories for predicted object. Our application will display the Result and Calories into our application.

CHALLENGES

- Recognizing the food item with the help of single picture
- Similar type of images for example roti and dosa, both are in same shape which we find difficult to recognize
- Dataset becomes much larger when it comes on food images, so currently we take a finite dataset for training

BENEFITS

- Precise and accurate recognition of food
- Rapid estimation of calorie helps users to monitor their nutritional intake
- Can keep track of dietary information

FUTURE WORK:

Food recognition & dietary intake estimation using computer vision is an emerging field of computer engineering. Our system has demonstrated identification or classification of food from food image using image processing & artificial neural network our system has demonstrated, decent accuracy in recognition of food. As automated food classification is an rapidly emerging field the techniques & systems have to adopt to the pace of development & improvement & add-ons to the system are sought. One of the most sought improvements is the addition of automatic calorie estimation depending upon the food type. Other improvement can be advent of a total dietary management system based upon proposed technique which can aid in selection of food types, nutrient cycles & can comment on food selection according to physio medical Requirements.

CONCLUSION:

As these enormous variety in food categories & classes of food items, & Hough intra class variations within every classes, automated food recognition using computer vision is on emerging challenge. With the advent of fitness devices & application & advancements in wearable devices, food recognition research is graining pace. The proposed system is an image processing based food recognition system with high accuracy & repetitive performance. Image processing is employed to segment the food position of the image & extract the food containing part of the image & then various
image parameters of the region of interest are computed. The parameter matrix is fed to a trained artificial neural network which classified the food in a particular type. Multiple methods such as surface feature extraction, Bag of
shapes & HSV background elimination have been employed
for food area segmentation. Morphology & binary image labelling have been used to obtain the various image
parameters. Levenberg marquardt function fitting neural network is used to approximate or classify the food type.
Combination of the above techniques yield higher accuracy as compared to previous methods.