

RECIPE RECOGNITION WITH DEEP LEARNING & WATSON

A PROJECT REPORT

Submitted by

**KUMUD JAIN
19BCE10217**

*in partial fulfillment for the award of the degree
of*

**BACHELOR OF TECHNOLOGY
in
PROGRAM OF STUDY**

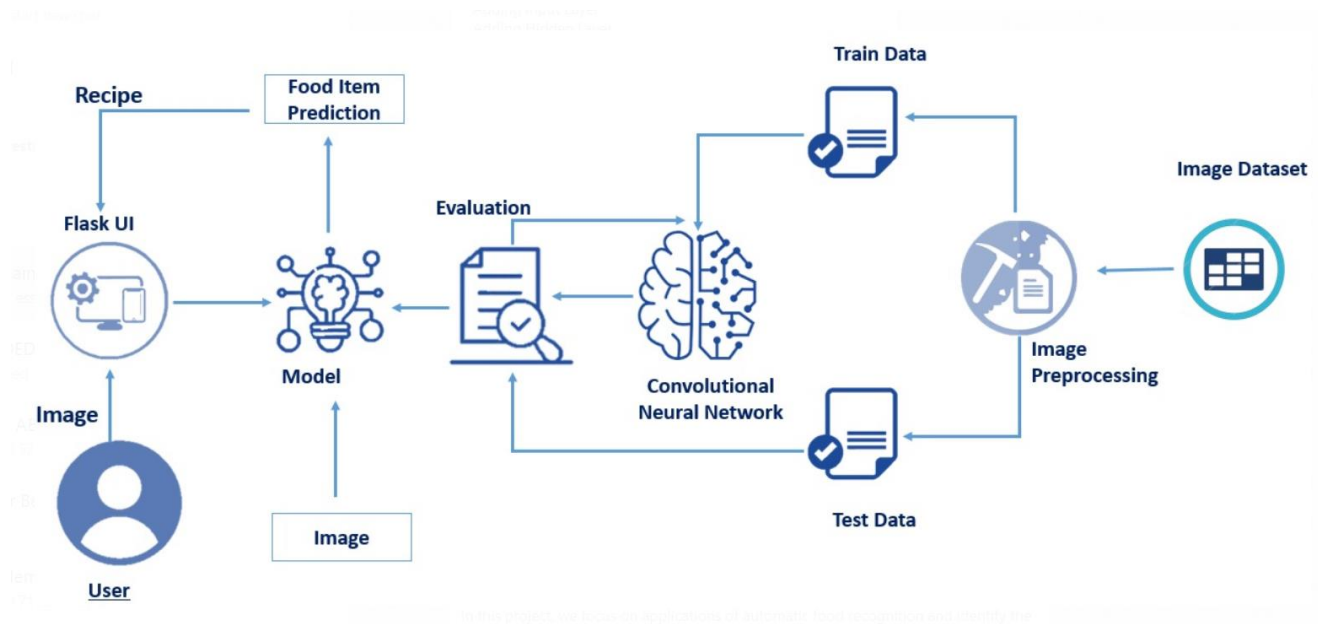


**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
VIT BHOVAL UNIVERSITY
KOTHRIKALAN, SEHORE
MADHYA PRADESH - 466114**

JAN 2022

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
1	INTRODUCTION	3
2	PROJECT OBJECTIVE	4
3	PROJECT STRUCTURE	5
4	THEORETICAL ANALYSIS	6
5	CONCLUSION	22



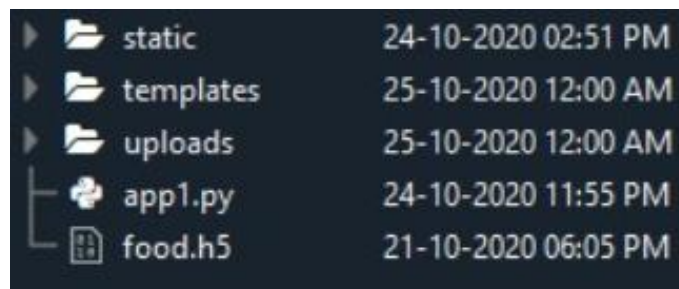
PROJECT OBJECTIVE

- Data Collection
 - Collect the dataset or Create the dataset
 - Data Preprocessing.
 - Import the ImageDataGenerator library
 - Configure ImageDataGenerator class
 - Apply ImageDataGenerator functionality to Trainset and Testset
- Model Building
 - Import the model building Libraries
 - Initializing the model
 - Adding Input Layer
 - Adding Hidden Layer
 - Adding Output Layer
 - Configure the Learning Process
 - Training the model
 - Save the Model

- Test the Model
- Application Building
 - Create an HTML file
 - Build Python Code

PROJECT STRUCTURE

Create a Project folder that contains files as shown below:



We are building a Flask Application which needs HTML pages stored in the templates folder and a python script app.py for server-side scripting. We need the model, which is saved, and the saved model in this content is food.h5. The static folder will contain JS and CSS files. Whenever we upload an image to predict, that image are saved in the uploads folder.

THEORETICAL ANALYSIS

DATA COLLECTION

ML or DL depends heavily on data, without data, it is impossible for a machine to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.

You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository, etc. As of now, you can download it in the next step.

DOWNLOAD THE DATASET

The dataset used for this project was obtained from Kaggle. Please refer to the link given below to download the data set click here

https://drive.google.com/drive/folders/1K8T3_OQz1Ri42LceZPAs4eQBU-y2DiJP

The dataset contains three classes:

1. French Fries
2. Pizza
3. Samosa

IMAGE PREPROCESSING

Image Pre-processing includes the following main tasks

1. Import ImageDataGenerator Library.
2. Configure ImageDataGenerator Class.
3. Applying ImageDataGenerator functionality to the trainset and test set.

Note: The ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data.

IMPORT ImageDataGenerator LIBRARY

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np#used for numerical analysis
```

CONFIGURE ImageDataGenerator CLASS

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

There are five main types of data augmentation techniques for image data; specifically:

1. Image shifts via the `width_shift_range` and `height_shift_range` arguments.
2. Image flips via the `horizontal_flip` and `vertical_flip` arguments.
3. Image rotations via the `rotation_range` argument
4. Image brightness via the `brightness_range` argument.
5. Image zoom via the `zoom_range` argument.

An instance of the `ImageDataGenerator` class can be constructed.

```
#setting parameter for Image Data agumentation to the traing data
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
```

Apply ImageDataGenerator Functionality To Trainset And Testset

Let us apply `ImageDataGenerator` functionality to `Trainset` and `Testset` by using the following code for the Training set using the `flow_from_directory` function.

Arguments:

directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.

batch_size: Size of the batches of data. Default: 32.

target_size: Size to resize images to after they are read from disk.

class_mode:

1. 'int': means that the labels are encoded as integers (e.g. for `sparse_categorical_crossentropy` loss).

2. 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).
3. 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
4. None (no labels).

There are 2400 images belonging to 3 classes (French fries, Pizza, and Samosa)

For Testset

There are 600 images belonging to 3 classes (French fries, Pizza and Samosa)

validation_split: Optional float between 0 and 1, the fraction of data to reserve for validation.

```
#performing data agumentation to train data
x_train=train_datagen.flow_from_directory(directory=r'E:\FOOD Classification\Food-Classification-from-I
, target_size=(64,64), batch_size=32, class_mode='categorical')
```

```
#Image Data agumentation to the testing data
test_datagen=ImageDataGenerator(rescale=1./255)
```

```
#performing data agumentation to test data
x_test=test_datagen.flow_from_directory(directory=r'E:\FOOD Classification\Food-Classification-from-Ima
, target_size=(64,64), batch_size=32, class_mode='categorical')
```

MODEL BUILDING

The neural network model is to be built by adding different network layers like convolution, pooling, flattening, dropout, and neural layers.

In this milestone, we start building our model by:

1. Initializing the mode
2. Adding Convolution layers
3. Adding Pooling layers
4. Flatten layer
5. Full connection layers which include hidden layers

At last, we compile the model with layers we added to complete the neural network structure

INITIALIZING THE MODEL

A sequential model is a linear stack of layers. You can create a Sequential model by passing a list of layer instances to the constructor: from Keras. models import Sequential from Keras as follows.

```
# create model  
model=Sequential()
```

ADDING CNN LAYERS

For information regarding CNN Layers refer to the link

Link: <https://victorzhou.com/blog/intro-to-cnns-part-1/>

We are adding a convolution layer with activation function as “relu” and with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.

Max pool layer is used to downsample the input.

The dropout layer is used to deactivate the neurons randomly. Dropout(0.2) indicates that 20 % of the neurons are deactivated.

Flatten layer flattens the input. Does not affect the batch size.

```
# adding model layer
model.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))#convolutional layer
model.add(MaxPooling2D(pool_size=(2,2))) #MaxPooling2D-for downsampling the input

model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))#dropping input randomly for preventing from overfitting

model.add(Flatten())#flatten the dimension of the image
model.add(Dense(32))#deeply connected neural network layers.
```

ADDING DENSE LAYERS

The dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

```
model.add(Dense(3,activation='softmax'))#output layer with 3 neurons
```

Understanding the model is very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

```
model.summary()#summary of our model
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 62, 62, 32)	896

max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0

conv2d_1 (Conv2D)	(None, 29, 29, 32)	9248

max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0

flatten (Flatten)	(None, 6272)	0

dense (Dense)	(None, 32)	200736

dense_1 (Dense)	(None, 3)	99
=====		
Total params: 210,979		
Trainable params: 210,979		
Non-trainable params: 0		

CONFIGURE THE LEARNING PROCESS

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. Loss function is used to find error or deviation in the learning process. Keras requires loss function during the model compilation process.

Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Here we are using adam optimizer

Metrics are used to evaluate the performance of your model. It is similar to loss function, but not used in the training process

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

TRAIN THE MODEL

Now, let us train our model with our image dataset. fit generator functions used to train a deep-learning neural network

Arguments:

`steps_per_epoch`: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of `steps_per_epoch` as the total number of samples in your dataset divided by the batch size.

`Epochs` : an integer and number of epochs we want to train our model for.

Validation data can be either:

1. an inputs and targets list
2. a generator
3. an inputs, targets, and `sample_weights` list which can be used to evaluate
4. the loss and metrics for any model after any epoch has ended.

`validation_steps`: only if the `validation_data` is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
# Fit the model  
model.fit_generator(generator=x_train, steps_per_epoch = len(x_train),  
                    epochs=80, validation_data=x_test, validation_steps = len(x_test))
```

SAVE THE MODEL

The model is saved with .h5 extension as follows

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

```
# Save the model  
model.save('food.h5')
```

TEST THE MODEL

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.

Load the saved model using load_model

```
from tensorflow.keras.models import load_model  
from keras.preprocessing import image  
model = load_model("food.h5") #Loading the model for testing
```

Taking an image as input and checking the results

```
img = image.load_img(r"E:\FOOD Classification\Food-Classification-from-Images-Using-Convolutional-Neural
x = image.img_to_array(img)#image to array
x = np.expand_dims(x,axis = 0)#changing the shape
pred = model.predict_classes(x)#predicting the classes
pred
```

By using the model we are predicting the output for the given input image

```
index=['french_fries', 'pizza', 'samosa']
result=str(index[pred[0]])
result

'samosa'
```

APPLICATION BUILDING

After the model is built, we will be integrating it into a web application so that users can interact with the model.

CREATE HTML PAGES

We use HTML to create the front end part of the web page. Here, we created 4 html pages- about.html, base.html, index6.html, info.html.

- about.html displays the home page.
- Info.html displays all important details to be known about Malaria.
- base.html and index6.html accept input from the user and predicts the values.

We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

BUILD PYTHON CODE AND TEST THE OUTPUT

Let us build flask file 'app.py' which is a web framework written in python for server-side scripting. Let's see step by step procedure for building the backend application.

The app starts running when “_name_” constructor is called in main.

render_template is used to return HTML file.

“GET” method is used to take input from the user.

“POST” method is used to display the output to the user.

- Importing Libraries

```
import os
import numpy as np #used for numerical analysis
from flask import Flask,request,render_template#Flask-It is our framework which
#we are going to use to run/serve our application.
#request-for accessing file which was uploaded by the user on our application.
#render_template- used for rendering the html pages
from tensorflow.keras.models import load_model#to load our trained model
from tensorflow.keras.preprocessing import image
```

- Routing to the HTML Page

```
app=Flask(__name__)#our flask app
model=load_model('food.h5')#loading the model

@app.route("/") #default route
def upload_file():
    return render_template("RR.html")#rendering html page

@app.route("/about") #route about page
def upoad_file1():
    return render_template("RR.html")#rendering html page

@app.route("/upload") # route for info page
def upload_file2():
    return render_template("RRP.html")#rendering html page
```

- Showcasing prediction on UI

- When the image is uploaded, it predicts whether the uploaded the image is French fries or pizza or samosa. If the image predicts value as 1, then it is displayed as “Pizza”. If the image predicts value as 0, then it is displayed as “French Fries”. If the image predicts value as 2, then it is displayed as “Samosa”.

```
@app.route("/predict",methods=["GET","POST"]) #route for our prediction
def upload():
    if request.method=='POST':
        f=request.files['file'] #requesting the file
        basepath=os.path.dirname('__file__')#storing the file directory
        filepath=os.path.join(basepath,"uploads",f.filename)#storing the file in uploads folder
        f.save(filepath)#saving the file

        img=image.load_img(filepath,target_size=(64,64)) #load and reshaping the image
        x=image.img_to_array(img)#converting image to array
        x=np.expand_dims(x,axis=0)#changing the dimensions of the image

        pred = model.predict_classes(x) # predicting classes
        print(pred) # printing the prediction
        index = ['French Fries', 'Pizza', 'Samosa']
        result = str(index[pred[0]])
        if (result=="French Fries"):
            return render_template("0.html",showcase = str(result))
        elif (result=="Pizza"):
            return render_template("1.html",showcase = str(result))
        else:
            return render_template("2.html",showcase = str(result))
        #return result#resturing the result
    else:
        return None

#port = int(os.getenv("PORT"))
if __name__=="__main__":
    app.run(debug=False)#running our app
    #app.run(host='0.0.0.0', port=8000,debug=False)
```

- Run The app in the local browser
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page

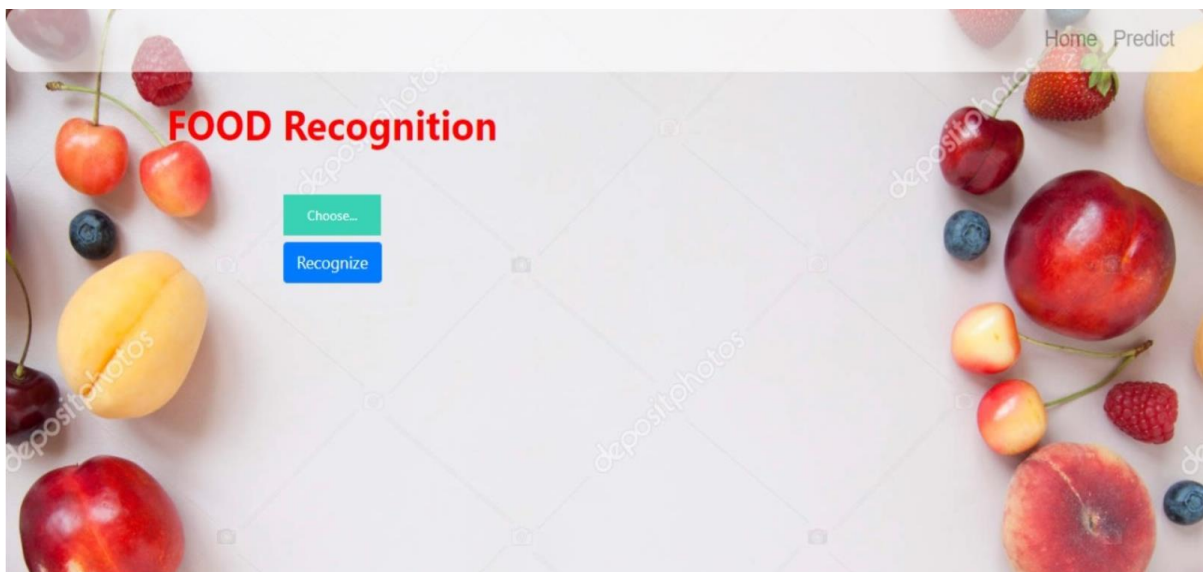
```
version
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- Then it will run on localhost:5000

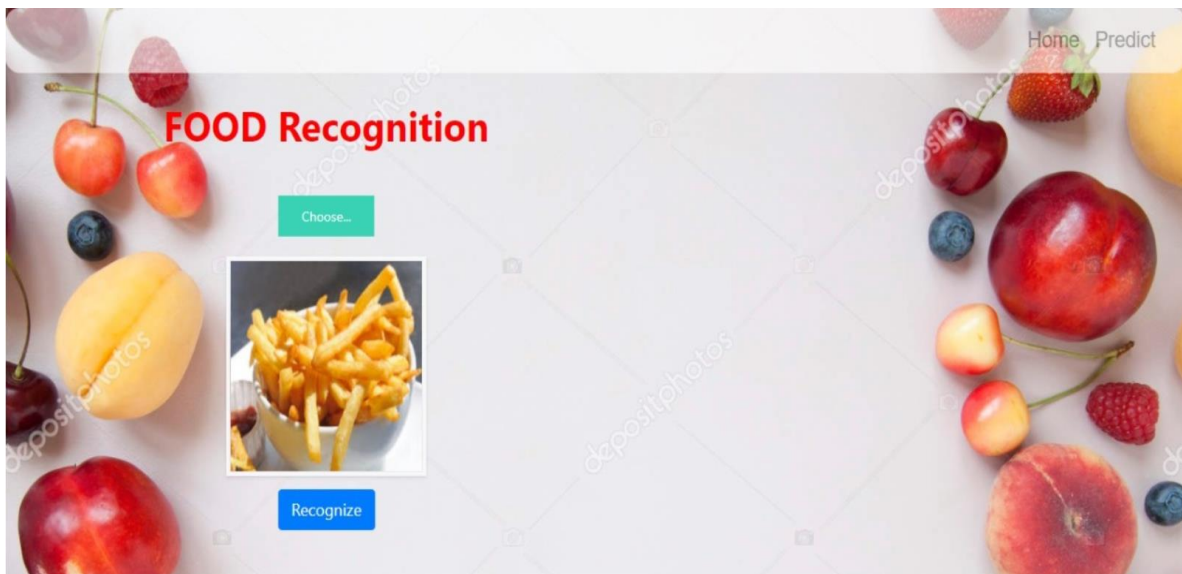
- Navigate to the localhost (<http://127.0.0.1:5000/>) where you can view your web page.
- Let's see how our about.html page looks like:



- When "predict" button is clicked it will redirect to prediction page

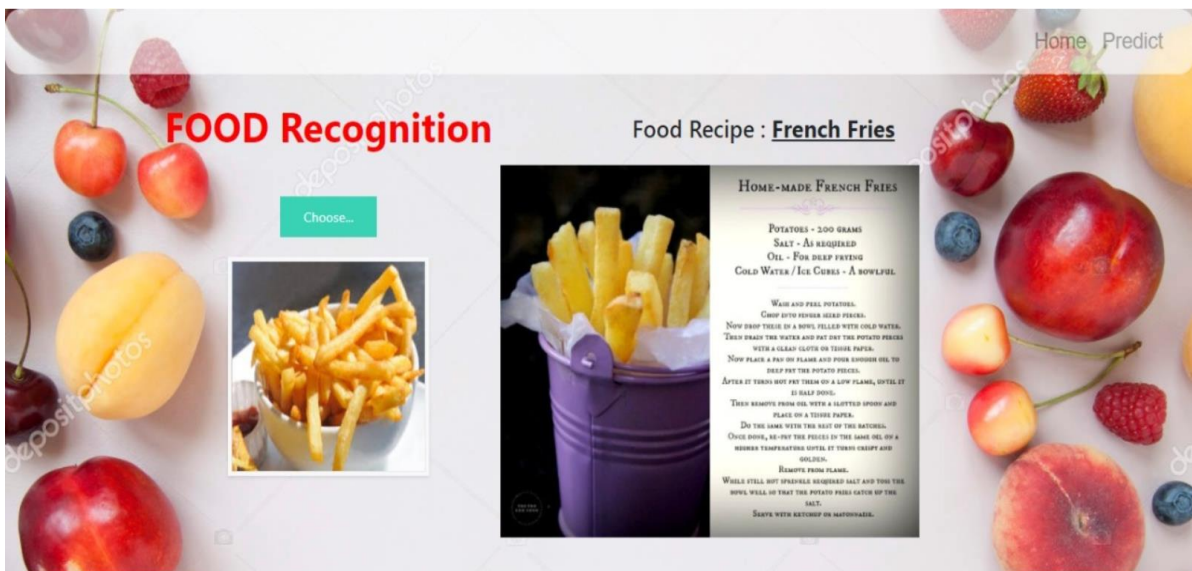


- In order to get the output one need to upload the image and click on recognize.



- Upload the image and click on Predict button to view the result on “RRP.html” page on localhost.
- Prediction-1:

If the uploaded image is French Fries the matching recipe for French Fries will be showcased.




- If the uploaded image is Pizza the matching recipe for Pizza will be showcased.

Home Predict

FOOD Recognition

Choose...



Food Recipe: Pizza


Student Recipe Card	
Name of Recipe French Bread Pizza	Difficulty Easy
Number of Servings 2	Method Slice French bread lengthways to create the base. Stir sauce ingredients together and spread over base. Arrange your chosen toppings on top; ham, mushrooms, etc. Spread the cheese on top and bake in the oven at 180 degrees for about 15 mins, until browned and cooked through.
Ingredients Pizza Sauce 1/2kg Tomato Puree 5 Tbsp Olive Oil 1 Tsp Dried Oregano 1 Tsp Dried Basil 0.5 Tsp Dried Rosemary 300 ml Water And... French Stick Pizza Toppings Grated Cheese	

- If the uploaded image is Samosa the matching recipe for Samosa will be showcased.

Home Predict

FOOD Recognition

Choose...



Food Recipe : Samosa

Crispy Chicken Keema Samosas Recipe

The Ingredients

- Chicken Mince - 500 gm
- Ginger Garlic Paste - 1/2 Cup
- Onion - 1/2 Cup (finely Chopped)
- Coriander Leaves - 1/2 Cup
- Spiced Tomato Puree - 1/2 Cup
- Spring Onion - 1/2 Cup
- Lemon Juice - 1/2 Cup
- Fresh Coriander Leaves - 1/2 Cup
- Salt - To Taste
- Turmeric - 1/2 Teaspoon
- Garam Masala - 1/2 Teaspoon (or 1/4 Cup)
- Samosa Shells - 10-12

Procedure

- Heat a small pan with 1 teaspoon oil. Add and put the chicken mince in it. Add ginger garlic paste salt and cook 5 minutes.
- Add onion puree (sauté), coriander leaves, and lemon juice. Add finely chopped onion. Let the mixture cook for 10 minutes and add spring onion to it.
- Take ready samosa shell and fill it with a triangular shape. Fill the same cooked mince in it. Lock the other edge with wet flour paste.
- Heat oil in a pan. Reduce the flame to medium and put the samosas in the oil. Deep fry the samosas until golden.
- Your Crispy Chicken Keema Samosas are ready to serve. Serve it with lemon.

CONCLUSION

From this project, we have successfully:

- Known fundamental concepts and techniques of Convolutional Neural Network.
- Gained a broad understanding of image data.
- Known how to pre-process/clean the data using different data pre-processing techniques.
- Known how to build a web application using Flask framework.