Configuration Manual

MSc Research Project
Data Analytics

Debratna Dhali
Student ID: 19201028

School of Computing
National College of Ireland

Supervisor: Dr. Majid Latifi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Debratna Dhali
Student ID:	19201028
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Majid Latifi
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	894
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Debratna Dhali
Date:	22nd September 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Debratna Dhali
19201028

1 Introduction

The configuration manual covers various aspects of the implementation process. This involves the hardware specification of the system on which the implementation was done along with the software required. It also covers the different stages of the implementation process and the overall evaluation of the research "Food Recommendation System Considering Calorie Estimated From Food Images UsingInceptionV3".

The subsequent sections shed more light on the implementation stages and computational requirements used to realise the project.

2 System Configuration

In this section the details about the hardware and software configurations used in this research is discussed.

2.0.1 Hardware Requirements

- Processor Intel(R) Core(TM) i5-8265U CPU @ 1.80GH
- RAM 8.00 GB
- System Type 64-bit Windows Operating System
- GPU Intel(R) UHD Graphics Family
- Storage 512 GB SSD

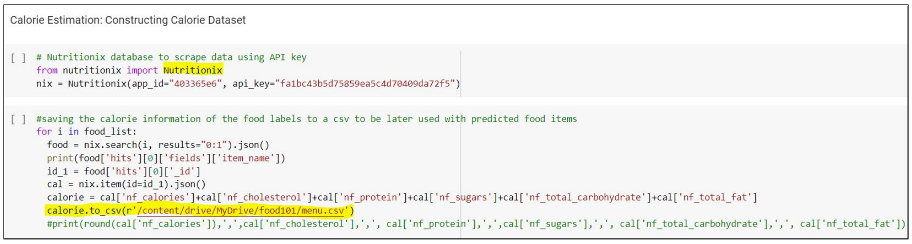
2.0.2 Software Requirements

- Microsoft Excel 2010: This is a microsoft product which is used to store the dataset which for this research is the calorie information dataset and the data is stored as csv format.
- Anaconda Distribution-Jupyter Notebook: It is an open source platform downloaded from the anaconda website. It houses a lot of design frameworks integrated into it such as Jupyter Notebook, Spyder, R Studio etc. The in-system Jupyter notebook was used mainly to split the original dataset into training and testing sets.

- Google Colab Pro: Google offers Colab which is a free cloud service having an integrated Jupyter notebook. Most of the implementation starting from exploratory data analysis, implementation of image classification model, constructing calorie dataset and recommender system to evaluation of implemented model has been done on Colab Pro. This service provides a RAM of 24 GB and GPUs either one of K80, P100, T4 based on usage.

3 Constructing Calorie Dataset

The calorie dataset was constructed by scraping the Nutritionix Database and storing the results in a csv file. In figure 1, the steps followed to implement this has been shown.


Calorie Estimation: Constructing Calorie Dataset

```
[ ] # Nutritionix database to scrape data using API key
    from nutritionix import Nutritionix
    nix = Nutritionix(app_id="403365e6", api_key="fa1bc43b5d75859ea5c4d70409da72f5")

[ ] #saving the calorie information of the food labels to a csv to be later used with predicted food items
    for i in food_list:
      food = nix.search(i, results="0:1").json()
      print(food['hits'][0]['fields']['item_name'])
      id_1 = food['hits'][0]['_id']
      cal = nix.item(id=id_1).json()
      calorie = cal['nf_calories']+cal['nf_cholesterol']+cal['nf_protein']+cal['nf_sugars']+cal['nf_total_carbohydrate']+cal['nf_total_fat']
      calorie.to_csv(r'/content/drive/MyDrive/food101/menu.csv')
      #print(round(cal['nf_calories']),',',cal['nf_cholesterol'],',', cal['nf_protein'],',',cal['nf_sugars'],',', cal['nf_total_carbohydrate'],',', cal['nf_total_fat'])
```

Figure 1: Calorie Dataset Construction

4 Dataset Description

The food image dataset to perform food detection was downloaded from [1].

This dataset contained food images from 101 different food labels. A total of 101,000 images are included in the dataset. Also, 750 images per class label were kept for training the data while 250 images per class were kept for testing the model. Figure 2 provides a sneak peak into the dataset used in the research.

Figure 2: Food Dataset

The second part of the research focusses on getting the calorie of the detected food item. Hence, a calorie dataset was constructed by scraping the Nutritionix database. This data was stored in a csv file for further use. Figure 3 shows the calorie data which has columns like "Item", "Calorie", "Cholesterol", "Protein", "Fat" etc.

[1]https://data.vision.ee.ethz.ch/cvl/datasets$_{e}xtra/food-101/$

Item	Serving Size	Calories	Total Fat	Cholesterol	Sodium	Carbohydr	Sugars	Protein
Macaron	100	404	13	0	247	72	71	3.6
Beignet	100	452	25	19	326	51	27	4.9
Samosa	100	262	17	27	423	24	1.6	3.5
Tiramisu	119	392	30	210	206	24	14	5.7
Tostada	100	148	8	33	387	13		7
Dumpling	52	120	2.8	1.8	440	20	0.8	3.2
knish	100	403	17.8	30	1040	53.5	3.4	7.1
croquette	70	105	3.9	22	249	14	0.9	4
couscous	100	112	0.2	0	5	23	0.1	3.8
porridge	100	50	0.2	0	6	11	0	1.4
seaweed_salad	93	106	7.3	0	1144	8	3.2	2.9
chow_mein	100	459	18	0	847	67	0.9	11
rigatoni	101	160	0.9	0	1	31	0.6	5.9
beef_tartare	224	552	44	340	403	5.7	1.8	33
cannoli	59	108	3.8	15	48	13	10	5.5
foie_gras	100	462	44	150	697	4.7		11
cupcake	100	305	3.7	0	413	67		4.3
ramen	100	436	16	0	2036	63	1.6	10
chicken_kiev	153	454	32.2	0	1.1	21.1	1.4	19.4
apple_pie	100	237	11	0	266	34		1.9
risotto	337	413	13	36	1451	54	4.7	14
fruitcake	100	324	9	5	101	62	27	2.9
chop_suey	464	582	30	86	790	48	7.5	31
scrambled_eggs	100	148	11	277	145	1.6	1.4	10
pizza	100	266	10	17	598	33	3.6	11
omelette	100	154	12	313	155	0.6	0.3	11
baby_back_ribs	242	668	45	176	531	13	11	48
baklava	76	306	20	21	213	29	16	5.5
beef_carpaccio	78	181	13	54	266	2	0.7	13

Figure 3: Calorie Dataset

5 Data Modelling for Food Classification

This section describes how the data was prepared before applying the model. Also, since transfer learning is used, the preparation for fine tuning the InceptionV3 model is also described in this section.

5.1 Dataset Sampling

The train and test dataset was sampled with 500 and 200 images from each category respectively. This was done because of computational boundaries. Figure 4 shows the sampling of training of images and figure 5 shows the sampling of testing images.

Sampling: Sampling train and test data. Using 500 images from each class for training and 200 images from each class for testing

```
[ ] # sampling for training dataset
    source = "/content/drive/MyDrive/food101/training_data"
    destination = "/content/drive/MyDrive/food101/train_sample"

    for food_item in range(len(food_list)):
      dir = os.path.join(source,food_list[food_item])
      os.chdir(dir)
      print("Copying images into",food_list[food_item])
      os.makedirs(os.path.join(destination, food_list[food_item]))
      for c in random.sample(glob.glob('*.jpg'),500):
        shutil.copy(c,os.path.join(destination, food_list[food_item]))
```

```
Copying images into apple_pie
Copying images into baby_back_ribs
Copying images into baklava
Copying images into beef_carpaccio
Copying images into beef_tartare
Copying images into beet_salad
Copying images into beignets
Copying images into bibimbap
Copying images into bread_pudding
Copying images into breakfast_burrito
Copying images into bruschetta
Copying images into caesar_salad
Copying images into cannoli
```

Figure 4: Sampling Training Data

```
[ ]  # sampling for test dataset
     source = "/content/drive/MyDrive/food101/test_data"
     destination = "/content/drive/MyDrive/food101/test_sample"

     for food_item in range(len(food_list)):
       dir = os.path.join(source,food_list[food_item])
       os.chdir(dir)
       print("Copying images into",food_list[food_item])
       os.makedirs(os.path.join(destination, food_list[food_item]))
       for c in random.sample(glob.glob('*.jpg'),200):
         shutil.copy(c,os.path.join(destination, food_list[food_item]))

     Copying images into french_onion_soup
     Copying images into french_toast
     Copying images into fried_calamari
     Copying images into fried_rice
     Copying images into frozen_yogurt
     Copying images into garlic_bread
     Copying images into gnocchi
     Copying images into greek_salad
     Copying images into grilled_cheese_sandwich
     Copying images into grilled_salmon
     Copying images into guacamole
     Copying images into gyoza
     Copying images into hamburger
     Copying images into hot_and_sour_soup
     Copying images into hot_dog
     Copying images into huevos_rancheros
     Copying images into hummus
     Copying images into ice_cream
     Copying images into lasagna
     Copying images into lobster_bisque
     Copying images into lobster_roll_sandwich
     Copying images into macaroni_and_cheese
```

Figure 5: Sampling Test Data

5.2 Image Pre-processing

In this research, image pre-processing was done using the ImageDataGenerator which is a Keras library. It pre-processed1 images in real time while the model is running. Figure 6 shows the implementation of data processing. As highlighted, the images are rescaled or normalized so that the pixels range from [0-1] instead of [0-255] and resized to size 299X299 before feeding into the model.

```
#augmentation configuration for both train and test datasets
generate_train_data = ImageDataGenerator(rescale=1. / 255,shear_range=0.2,
                                         zoom_range=0.2,horizontal_flip=True)
generate_test_data = ImageDataGenerator(rescale=1. / 255)

# this is a generator that will read pictures found in subfolers of 'training_directory'
# and 'test_directory' and indefinitely generate batches of augmented image data

train_generator = generate_train_data.flow_from_directory(training_directory,          # target directory
                                                          target_size=(img_height, img_width),  # desired image size
                                                          batch_size=batch_size,       # batch size
                                                          class_mode='categorical')    # since we use classification_crossentropy loss,
                                                                                       # we need classification labels

test_generator = generate_test_data.flow_from_directory(test_directory,target_size=(img_height, img_width),batch_size=batch_size,
                                                        class_mode='categorical')
```

Figure 6: Data Augmentation

5.3 Downloading Pre-trained InceptionV3 Model

The base model is downloaded with weights of the imagenet dataset. The model is further initialized by adding a global spatial average pooling layer, a fully connected layer and a logistic layer for 120 classes as shown in figure 7.

Pre-Trained Model: Downloading pre-trained inception v3 model

```
[ ] base_model = InceptionV3(weights='imagenet', include_top=False) # create the base pre-trained model
    x = base_model.output                      # add a global spatial average pooling layer
    x = GlobalAveragePooling2D()(x)            # adding a fully-connected layer
    x = Dense(120,activation='relu')(x)        # and a logistic layer for 120 classes
    x = Dropout(0.2)(x)
```

Figure 7: Downloading base InceptionV3 model

5.4 Fine Tuning InceptionV3

After downloading the base model, it was further fine tuned mainly to reduce overfitting by using L2 regularization and the lambda value was set to 0.005. The fine tuned model was then trained and compiled using SGD with a learning rate of 0.001 as shown in figure 8

Fine Tuning Inception Model

```
[ ] # L2 regularization to reduce overfitting.
    pred = Dense(number_of_labels,kernel_regularizer=regularizers.l2(0.005), activation='softmax')(x) #0.005 is the value of lambda
    model = Model(inputs=base_model.input, outputs=pred) #training the model
    model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy']) # compiling the model using SGD with learning rate 0.001

    /usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
      "The `lr` argument is deprecated, use `learning_rate` instead.")
```

Figure 8: Fine Tuning InceptionV3 model

5.5 Setting up Checkpoints and Model Implementation

While training deep neural networks, it is important to set up model checkpoints so that in case of interruptions, the model can be restarted or reused from the last saved point.

Setting Up Checkpoints to retrieve model in case of interruptions

```
[ ] model_checkpoint = ModelCheckpoint(filepath='/content/drive/MyDrive/food101/best_model.hdf5', verbose=1, save_best_only=True)
    logs = CSVLogger('/content/drive/MyDrive/food101/logs.log')
```

Figure 9: Model Checkpoint

Post this the final model was implemented for 42 epochs as shown in figure below.

6 Model Evaluation

The food classification model was mainly evaluated based on the accuracy and loss metrics. Multiple cases were examined to conclude the best performing model as shown in 10, 11. Finally, 12 was chosen as the final model configuration as it gave the highest accuracy.

```
Epoch 00009: val_loss improved from 4.24247 to 4.02802, saving model to /content/drive/MyDrive/best_model_10.hdf5
Epoch 10/10
404/404 [==============================] - 1431s 4s/step - loss: 4.1220 - accuracy: 0.2437 - val_loss: 3.8289 - val_accuracy: 0.3264

Epoch 00010: val_loss improved from 4.02802 to 3.82894, saving model to /content/drive/MyDrive/best_model_10.hdf5
```

Figure 10: Loss and Accuracy on 10 Epochs for 101 Categories

```
Epoch 00014: val_loss improved from 2.06707 to 1.98278, saving model to /content/food-101/best_model_3class.hdf5
Epoch 15/15
480/480 [==============================] - 837s 2s/step - loss: 2.3318 - accuracy: 0.5294 - val_loss: 1.8887 - val_accuracy: 0.6446

Epoch 00015: val_loss improved from 1.98278 to 1.88866, saving model to /content/food-101/best_model_3class.hdf5
```

Figure 11: Loss and Accuracy Plots on 15 Epochs for 72 Categories

```
Epoch 00042: val_loss improved from 1.20476 to 1.18934, saving model to /content/drive/MyDrive/best_model_3class.hdf5
Epoch 43/45
480/480 [==============================] - 784s 2s/step - loss: 1.2070 - accuracy: 0.7830 - val_loss: 1.1809 - val_accuracy: 0.
7786

Epoch 00043: val_loss improved from 1.18934 to 1.18088, saving model to /content/drive/MyDrive/best_model_3class.hdf5
Epoch 44/45
480/480 [==============================] - 790s 2s/step - loss: 1.1899 - accuracy: 0.7880 - val_loss: 1.1748 - val_accuracy: 0.
7792

Epoch 00044: val_loss improved from 1.18088 to 1.17479, saving model to /content/drive/MyDrive/best_model_3class.hdf5
Epoch 45/45
480/480 [==============================] - 787s 2s/step - loss: 1.1705 - accuracy: 0.7917 - val_loss: 1.1694 - val_accuracy: 0.
7803

Epoch 00045: val_loss improved from 1.17479 to 1.16938, saving model to /content/drive/MyDrive/best_model_3class.hdf5
```

Figure 12: Loss and Accuracy on 45 Epochs for 72 Categories

The metrics were plotted so as to understand the accuracy and loss based per epoch and the code implemented to do that has been shown in figure 13.

Plotting the accuracy and loss graph

```
[ ]  print(fit.fit.keys())
     #  "Accuracy"
     plt.plot(fit.fit['acc'])
     plt.plot(fit.fit['val_acc'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'validation'], loc='upper left')
     plt.show()
     # "Loss"
     plt.plot(fit.fit['loss'])
     plt.plot(fit.fit['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'validation'], loc='upper left')
     plt.show()
```

Figure 13: Model Evaluation Code

The evaluation output of the InceptionV3 model is shown in figure 14

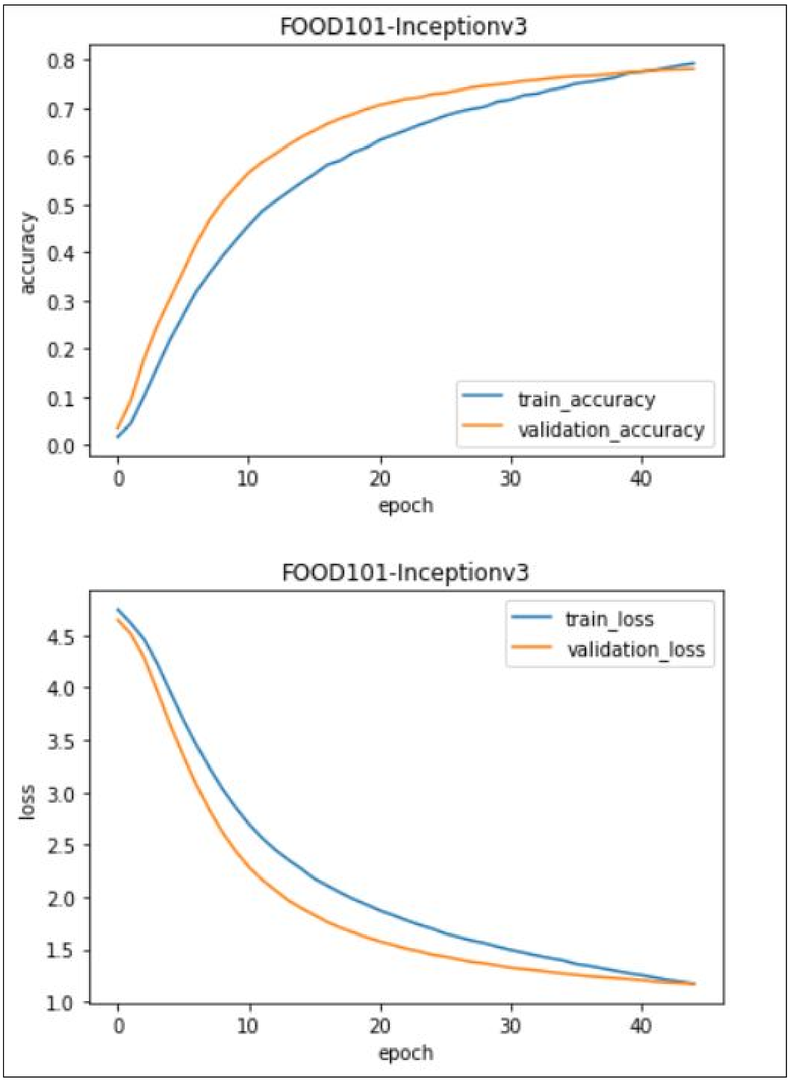


Figure 14: Model Evaluation Of InceptionV3 Model

7 Model Validation

To validate the model, random images from the internet were chosen and feeded into the network. Then they were visualized to see the final output of the system. The output is seen in figure below gnocchi

```
 Calories    Fat    Cholesterol    Sodium    Carbohydrate    Protein
----------  -----  -------------  --------  --------------  ---------
       254    1.5             37        18              52        7.7
Recommended Products : cannoli gnocchi prime_rib
```

Figure 15: Output 1: Model Correctly Classifying Gnocchi

fried_rice

```
 Calories    Fat    Cholesterol    Sodium    Carbohydrate    Protein
----------  -----  -------------  --------  --------------  ---------
       163    2.3             23       396              31        4.7
Recommended Products : gnocchi rigatoni Fruit 'n Yogurt Parfait
```

Figure 16: Output 2: Model Correctly Classifying Fried Rice