Configuration Manual

MSc Research Project
Data Analytics

Dhanashree Subhash Rane
Student ID: x20142498

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Dhanashree Subhash Rane…………………………………………………

Student ID: ………x20142498……………………………………………………………………..…….

Programme: ……MSc in Data Analytics…………………… **Year:** ……2021………..

Module: ……MSc Research Project …………………………………………………….……….

Lecturer: ……Dr.Chiristian Horn……………………………………………………………..………

Submission Due Date: ………31/01/2022..………………………………………………………………….………

Project Title: Improving food classification rate using Transfer learning methods ……………………………………………………………..……………………………………….………

Word Count: ………1264……………………… **Page Count:** ……………18………….…..……..………

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: ……Dhanashree Subhash Rane…………………………………………………

Date: ……31/01/2022……...………………………………………………………

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

Dhanashree Subhash Rane
Student ID: x20142498

1 Introduction

The goal of creating this document is to showcase the project's implementation in a succinct and systematic manner that may be copied if necessary. The goal of this research was to develop a model for classification of Food image utilizing a thorough methodological approach that included data cleaning and pre-processing. These manual details the tools and procedures used in the project.

2 System Specifications

The hardware requirements for running the experiment and smoothly executing code are listed below.

Operating System	MacOS
RAM	8.0 GB
Har Disk Space	100 GB Minimum
Processor	1.8GHz dual-core Intel Core i5, Turbo Boost up to 2.9GHz, with 3MB shared L3 cache

3 Tools/Technology

The Python programming language was used to build this project, along with an Integrated Development Environment (IDE) called Jupyter Notebook, which runs on the Anaconda platform. Below are the precise versions of the corresponding platform/language.

Programming Language	Python 3.8.3
IDE	Jupyter Notebook v. 6.0.3
Platform	Anaconda v. 4.9.2
Tools	Microsoft Excel, Overleaf, TeXstudio
Web Browser	Google Chrome

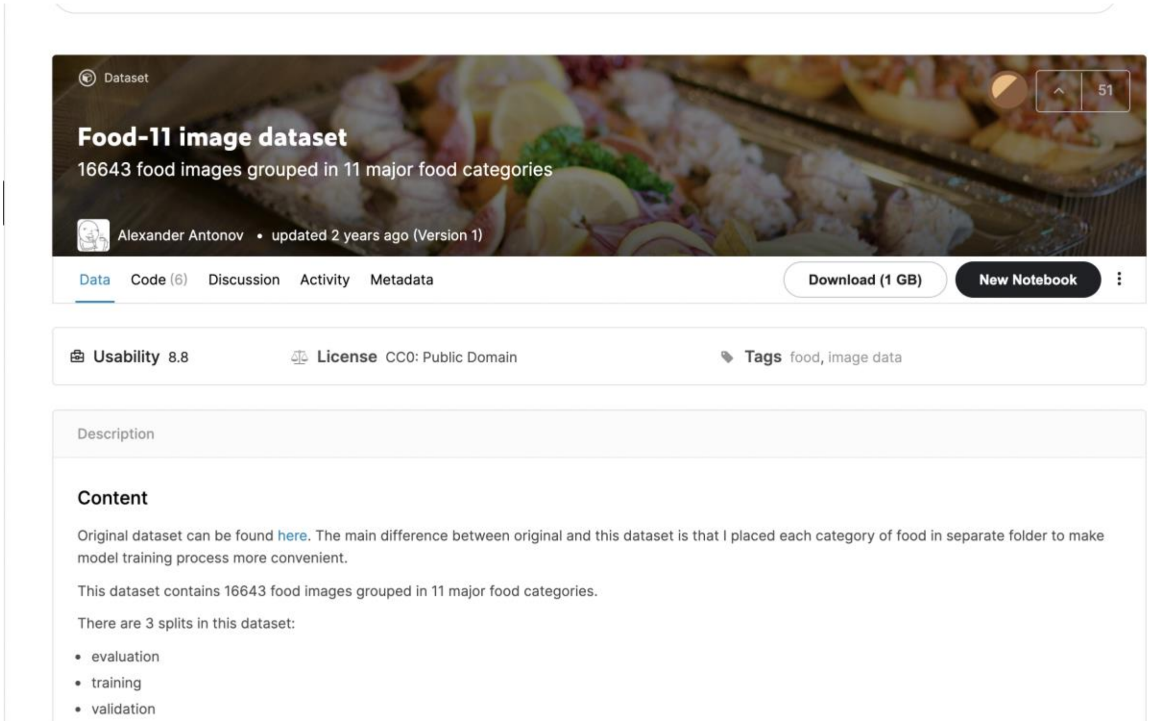
4 Pre-requisites software setup

The installation of the essential platform and languages is the first stage in completing this project.

- The link[1] is used to install Python.
- This link[2] was used to install Anaconda.
- After execution of the project, the results are visualized in the Jupyter Notebook using the libraries such as MatPlotLib, seaborn, and Plotly.

5 Data Collection

Data for this research is collected from repository[3].To Download the data (food-11 dataset) click on the download option shown in the below figure.



6 Implementation

The implementation of the project is divided into different processes.

6.1 Data Preparation and Storage

- Extracting the CSV data file (food-11) from the Kaggle repository[3] to the local machine.
- Importing the libraries in Jupyter Notebook as shown below.

[1] https://www.python.org/downloads/release/python-383/
[2] https://anaconda.org/conda-forge/conda/files?version=4.9.2
[3] https://www.kaggle.com/trolukovich/food11-image-dataset?select=evaluation

In [17]:

```python
import cv2
import os
import numpy as np
from PIL import Image
import pandas as pd
import pickle
import matplotlib.pyplot as plt
import plotly.graph_objects as go
```

In [93]:

```python
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

In [2]:

```python
# Importing Base class Tensorflow
import tensorflow as tf
```

In [3]:

```python
print("Tensorflow Version => ",tf.__version__)
```

```
Tensorflow Version =>  2.5.0
```

6.2 Data Visualisation

It is critical to understand the data and be aware of the features to focus on when performing data cleaning and pre-processing before beginning any data cleaning or transformation.

0.1 Data Visualisation ¶

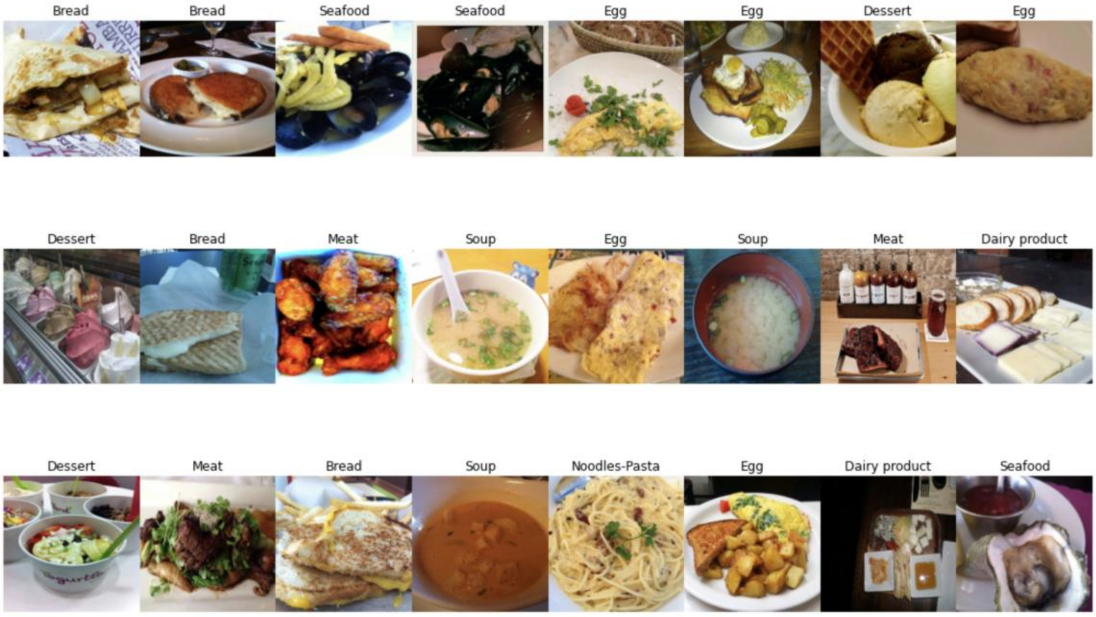
In [4]:

```python
train_dir = "training"
val_dir = "validation"
eval_dir = "evaluation"

target_size = (192,192,3)
Epochs = 50

precision = tf.keras.metrics.Precision(top_k=2)
recall = tf.keras.metrics.Recall(top_k=2)
```

In [5]:

```python
images_names = []
label_names = []

for label in os.listdir(train_dir):
    for image in os.listdir("{}/{}".format(train_dir,label)):
        images_names.append("{}/{}/{}".format(train_dir,label,image))
        label_names.append(label)
```

The above figure shows the Data has been divided into three parts which is training, validation and evaluation.



6.3 Data Augmentation

Data augmentation is important if we are dealing with image data in order to increase the amount of data by adding modified images with some small modification in order to balance the data. Here, we divided our work into two parts, We have processed data without augmentation and data with augmentation to compare both results before further implementation.

1.1.1 Data Generator Without Augmentation

In [7]:

```
# Data with no preprocessing
img_gen_no_aug = tf.keras.preprocessing.image.ImageDataGenerator()

train_data_no_aug = img_gen_no_aug.flow_from_directory(directory=train_dir,
                                    target_size=target_size[0:2],
                                    batch_size=4,
                                   )
val_data_no_aug  = img_gen_no_aug.flow_from_directory(directory=val_dir,
                                    target_size=target_size[0:2],
                                    batch_size=1,
                               )
```

```
Found 9866 images belonging to 11 classes.
Found 3430 images belonging to 11 classes.
```

1.1.2 Data Generator With Augmentation

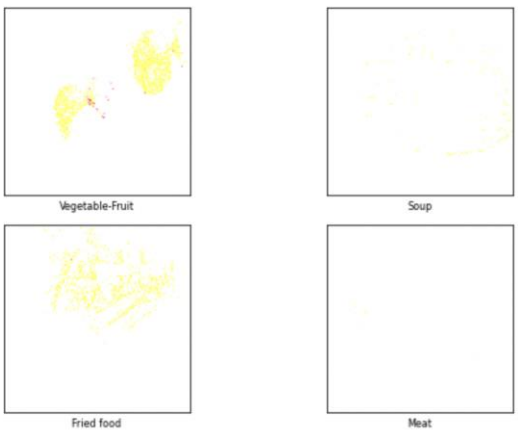
```
In [39]:
# Data with  preprocessing
img_gen_with_aug = tf.keras.preprocessing.image.ImageDataGenerator(rotation_range = 40,
                                                                   horizontal_flip=True,
                                                                   shear_range = 0.2,
                                                                   zoom_range = 0.2,
                                                                   rescale=1./255.0,)

# we will not do data augmentation with validation data
# As we want to make model perform well in real life
val_img_gen_with_aug = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1/255)


train_data_with_aug = img_gen_with_aug.flow_from_directory(directory=train_dir,
                            target_size=target_size[0:2],
                            batch_size=4,
                           )
val_data_with_aug  = val_img_gen_with_aug.flow_from_directory(directory=val_dir,
                            target_size=target_size[0:2],
                            batch_size=1,
                         )
```

```
Found 9866 images belonging to 11 classes.
Found 3430 images belonging to 11 classes.
```

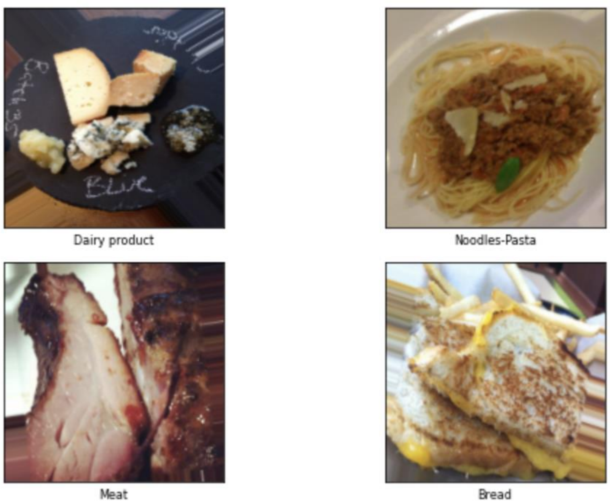
Below are the output of Data without augmentation and Data with augmentation.

```
In [11]:
# Images generated without augmentation
plot(train_data_no_aug.next(),(2,2),classes_,"No Augmentatioon")
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1]
Clipping input data to the valid range for imshow with RGB data ([0..1]
Clipping input data to the valid range for imshow with RGB data ([0..1]
Clipping input data to the valid range for imshow with RGB data ([0..1]
```




```
In [40]:
# Images generated with augmentation
plot(train_data_with_aug.next(),(2,2),classes_)
```



6.4 Data Modelling

Building a model is the most important step in the data mining process. In this research we have implemented three models of transfer learning, MobileNetV2, InceptionV3 and Custom CNN. The required libraries for the models have been already imported at the time of implementing each model.

6.4.1 MobileNetV2

Below figure shows the implementation of the MobileNetv2 model. We have performed training on both Data which is Data without augmentation and Data with augmentation in order to compare the results.

1.2 MobileNetV2

In [13]:
```
# Defining model
# We will be training from scratch
model = tf.keras.applications.MobileNetV2(include_top=False,weights="imagenet", input_shape=target_size)

model_MobileNetV2 = tf.keras.models.Sequential()
model_MobileNetV2.add(model)
model_MobileNetV2.add(tf.keras.layers.Flatten())

model_MobileNetV2.add(tf.keras.layers.Dense(256, activation='relu'))
model_MobileNetV2.add(tf.keras.layers.Dropout(0.5))
model_MobileNetV2.add(tf.keras.layers.BatchNormalization())


model_MobileNetV2.add(tf.keras.layers.Dense(len(classes_), activation='softmax'))

model_MobileNetV2.layers[0].trainable=False

```

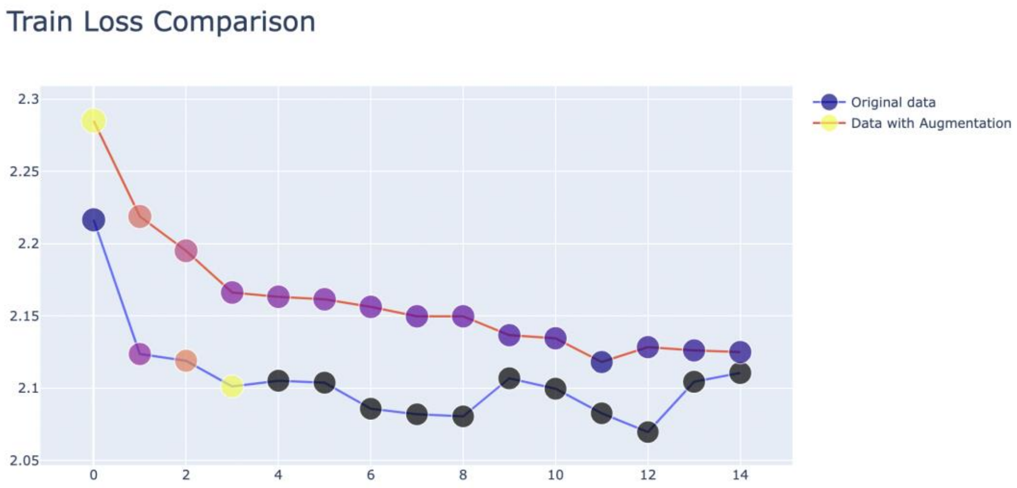
Below figure shows the implementation of comparison for Train loss and Test Accuracy

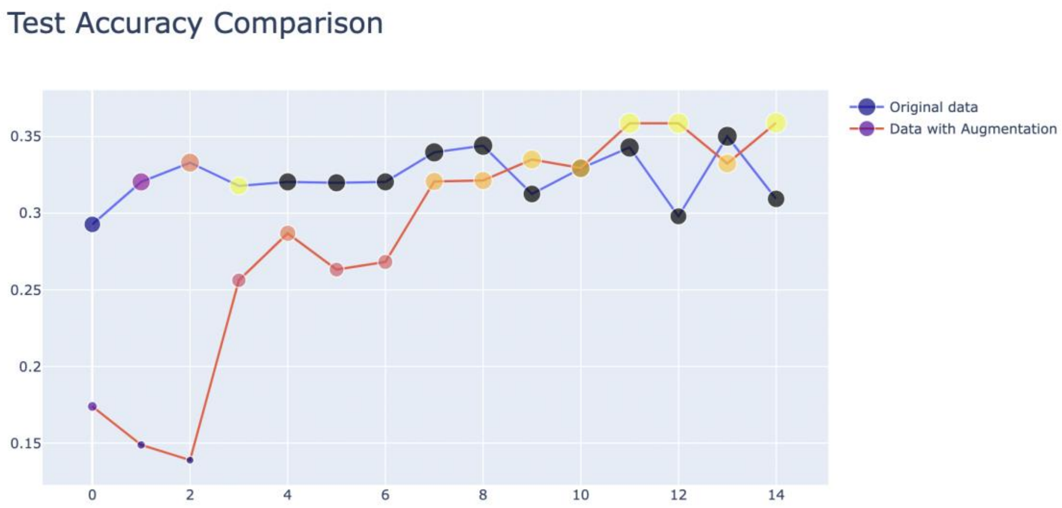
comparison

In [77]:
```
fig = go.Figure(data=go.Scatter(
    y=history_MobileNetV2_noAug.history["loss"],
    mode='lines+markers',
    marker=dict(size=np.array(history_MobileNetV2_noAug.history["loss"])*10,
                color=[0, 1, 2, 3]),
    name="Original data"
))


fig.add_trace(go.Scatter(
    y=history_MobileNetV2_Aug.history["loss"],
    mode='lines+markers',
    marker=dict(size=np.array(history_MobileNetV2_Aug.history["loss"])*10,
                color=np.array(history_MobileNetV2_Aug.history["loss"])*10),
    name="Data with Augmentation"
))


fig.update_layout(title=go.layout.Title(text="Train Loss Comparison",
                                        font=go.layout.title.Font(size=25)))

fig.show()
```

In [22]:
```
fig = go.Figure(data=go.Scatter(
    y=history_MobileNetV2_noAug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_MobileNetV2_noAug.history["val_accuracy"])*50,
                color=[0, 1, 2, 3]),
    name="Original data"
))


fig.add_trace(go.Scatter(
    y=history_MobileNetV2_Aug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_MobileNetV2_Aug.history["val_accuracy"])*50,
                color=np.array(history_MobileNetV2_Aug.history["val_accuracy"])*50),
    name="Data with Augmentation"
))


fig.update_layout(title=go.layout.Title(text="Test Accuracy Comparison",
                                        font=go.layout.title.Font(size=25)))

fig.show()
```

Below two figures shows training loss for both Data using MobileNetv2 and accuracy comparison.





6.4.2 Inceptionv3

Below figure shows the implementation of the Inceptionv3 model. We have performed training on both Data which is Data without augmentation and Data with augmentation in order to compare the results.

1.3 Inceptionv3

In [33]:
```
# Defining model
# We will be training from scratch
model = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet', input_shape=target_size)

model_InceptionV3 = tf.keras.models.Sequential()
model_InceptionV3.add(model)
model_InceptionV3.add(tf.keras.layers.Flatten())
model_InceptionV3.add(tf.keras.layers.BatchNormalization())
model_InceptionV3.add(tf.keras.layers.Dense(256, activation='relu'))
model_InceptionV3.add(tf.keras.layers.Dropout(0.5))
model_InceptionV3.add(tf.keras.layers.BatchNormalization())
model_InceptionV3.add(tf.keras.layers.Dense(128, activation='relu'))
model_InceptionV3.add(tf.keras.layers.Dropout(0.5))
model_InceptionV3.add(tf.keras.layers.BatchNormalization())
model_InceptionV3.add(tf.keras.layers.Dense(len(classes_), activation='softmax'))

model_InceptionV3.layers[0].trainable=False
```

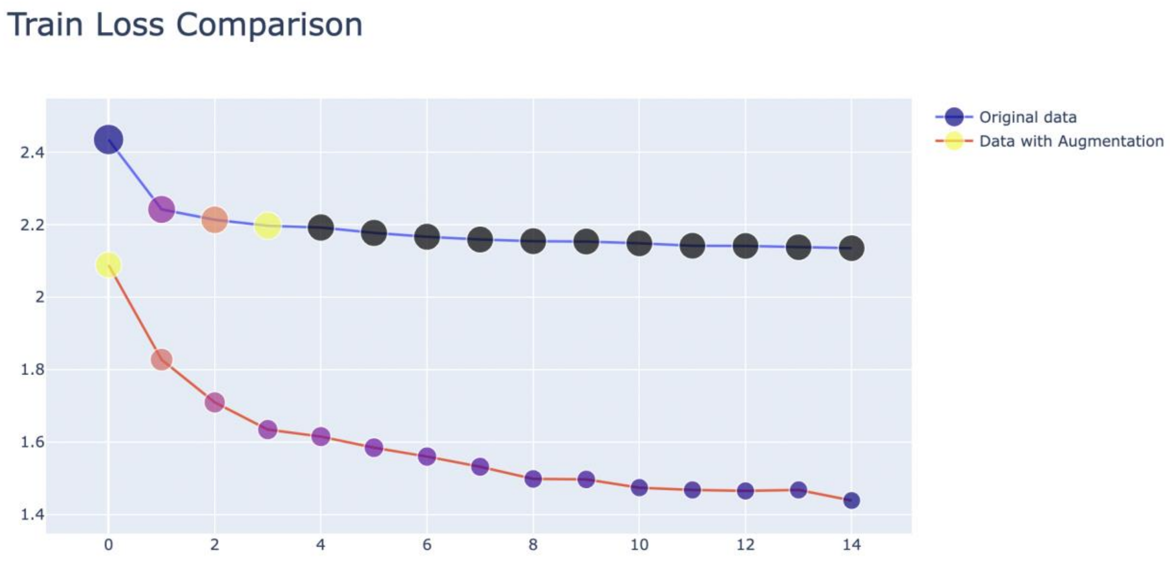
Below figure shows the implementation of comparison for Train loss and Test Accuracy.

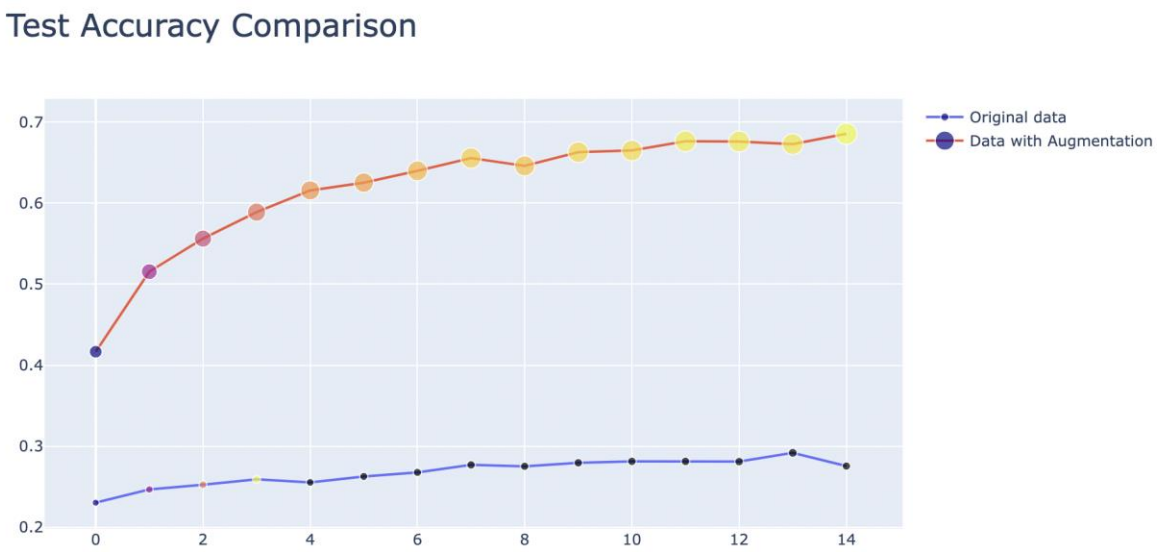
Comparison

In [78]:
```
fig = go.Figure(data=go.Scatter(
    y=history_model_InceptionV3_noAug.history["loss"],
    mode='lines+markers',
    marker=dict(size=np.array(history_model_InceptionV3_noAug.history["loss"])*10,
                color=[0, 1, 2, 3]),
    name="Original data"
))


fig.add_trace(go.Scatter(
    y=history_InceptionV3_Aug.history["loss"],
    mode='lines+markers',
    marker=dict(size=np.array(history_InceptionV3_Aug.history["loss"])*10,
                color=np.array(history_InceptionV3_Aug.history["loss"])*10),
    name="Data with Augmentation"
))


fig.update_layout(title=go.layout.Title(text="Train Loss Comparison",
                                        font=go.layout.title.Font(size=25)))

fig.show()
```

In [45]:
```
fig = go.Figure(data=go.Scatter(
    y=history_model_InceptionV3_noAug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_model_InceptionV3_noAug.history["val_accuracy"])*25,
                color=[0, 1, 2, 3]),
    name="Original data"
))


fig.add_trace(go.Scatter(
    y=history_InceptionV3_Aug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_InceptionV3_Aug.history["val_accuracy"])*25,
                color=np.array(history_InceptionV3_Aug.history["val_accuracy"])*25),
    name="Data with Augmentation"
))


fig.update_layout(title=go.layout.Title(text="Test Accuracy Comparison",
                                        font=go.layout.title.Font(size=25)))

fig.show()
```

Below two figures shows training loss for both Data using Inceptionv3 and accuracy comparison.





6.4.3 Custom Architecture

Below figure shows the implementation of the Inceptionv3 model. We have performed training on both Data which is Data without augmentation and Data with augmentation in order to compare the results.

1.4 Custom Architechture

In [49]:
```
custom_model = tf.keras.models.Sequential()

custom_model.add(tf.keras.layers.Conv2D(64, (3, 3), padding='same', input_shape=(target_size[0],target_size[1],3),
custom_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

custom_model.add(tf.keras.layers.Conv2D(128, (3, 3), padding='same',activation='relu'))
custom_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

custom_model.add(tf.keras.layers.Conv2D(256, (3, 3), padding='same',activation='relu'))
custom_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
custom_model.add(tf.keras.layers.BatchNormalization())


custom_model.add(tf.keras.layers.Conv2D(512, (3, 3), padding='same',activation='relu'))
custom_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

custom_model.add(tf.keras.layers.Flatten())

custom_model.add(tf.keras.layers.Dense(1024, activation='relu',input_dim=128))
custom_model.add(tf.keras.layers.Dropout(0.3))
custom_model.add(tf.keras.layers.BatchNormalization())

custom_model.add(tf.keras.layers.Dense(128, activation='relu'))

custom_model.add(tf.keras.layers.Dense(len(classes_), activation='softmax'))
```

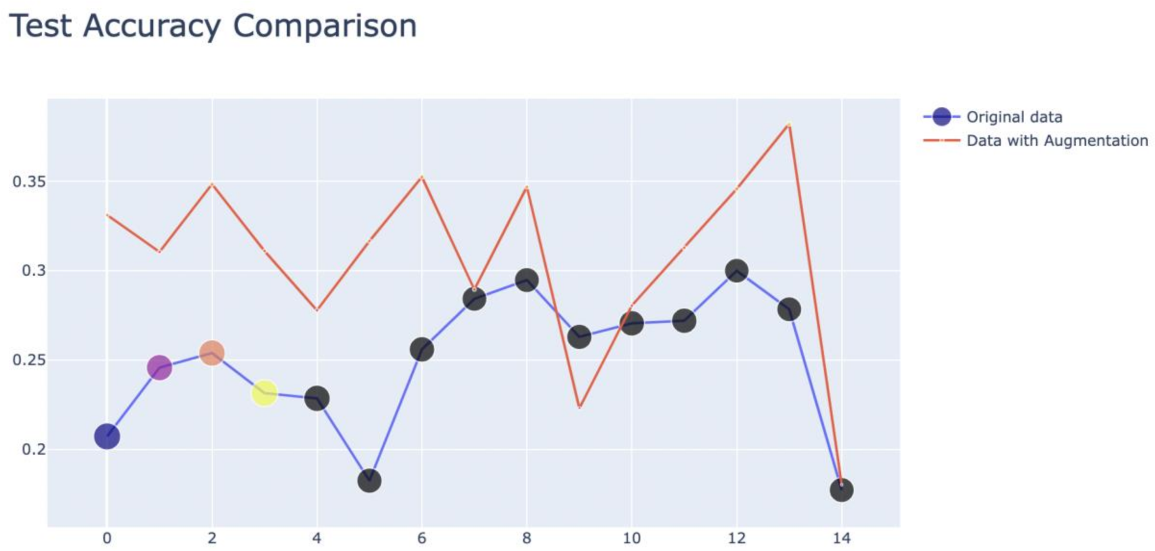
Below figure shows the implementation of comparison for Train loss and Test Accuracy.

Comparison

In [79]:
```
fig = go.Figure(data=go.Scatter(
    y=history_model_custom_noAug.history["loss"],
    mode='lines+markers',
    marker=dict(size=np.array(history_model_custom_noAug.history["loss"])*10,
                color=[0, 1, 2, 3]),
    name="Original data"
))


fig.add_trace(go.Scatter(
    y=history_model_custom_Aug.history["loss"],
    mode='lines+markers',
    marker=dict(size=np.array(history_model_custom_Aug.history["loss"])*10,
                color=np.array(history_model_custom_Aug.history["loss"])*10),
    name="Data with Augmentation"
))


fig.update_layout(title=go.layout.Title(text="Train Loss Comparison",
                                        font=go.layout.title.Font(size=25)))

fig.show()
```

In [62]:
```
fig = go.Figure(data=go.Scatter(
    y=history_model_custom_noAug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_model_custom_noAug.history["loss"])*10,
                color=[0, 1, 2, 3]),
    name="Original data"
))


fig.add_trace(go.Scatter(
    y=history_model_custom_Aug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_model_custom_Aug.history["val_accuracy"])*10,
                color=np.array(history_model_custom_Aug.history["val_accuracy"])*10),
    name="Data with Augmentation"
))


fig.update_layout(title=go.layout.Title(text="Test Accuracy Comparison",
                                        font=go.layout.title.Font(size=25)))

fig.show()
```

Below two figures shows training loss for both Data using Inceptionv3 and accuracy comparison.

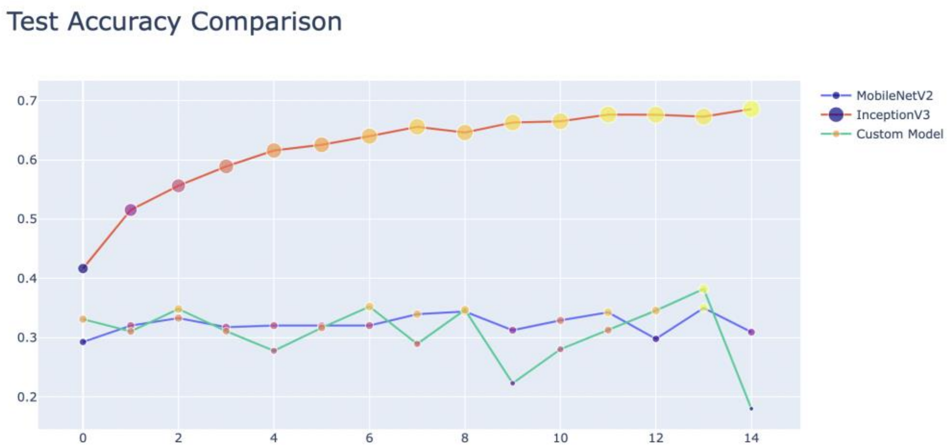


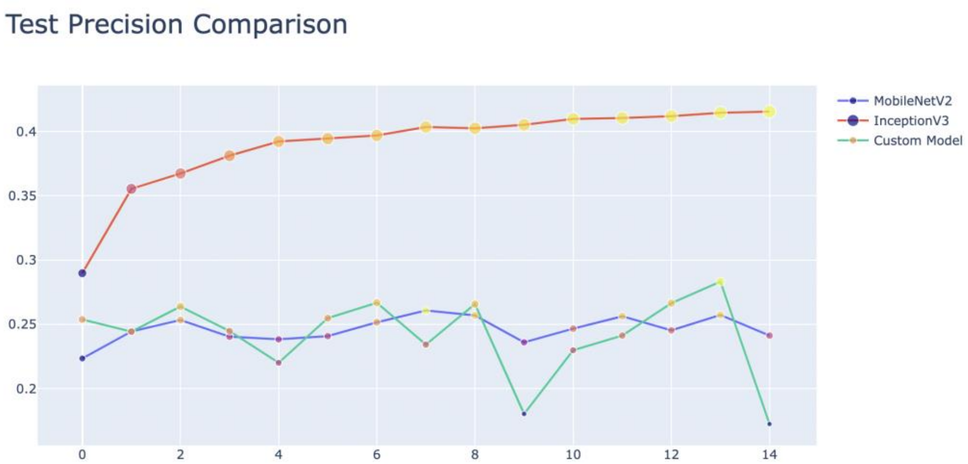
6.4.4 Complete model comparison

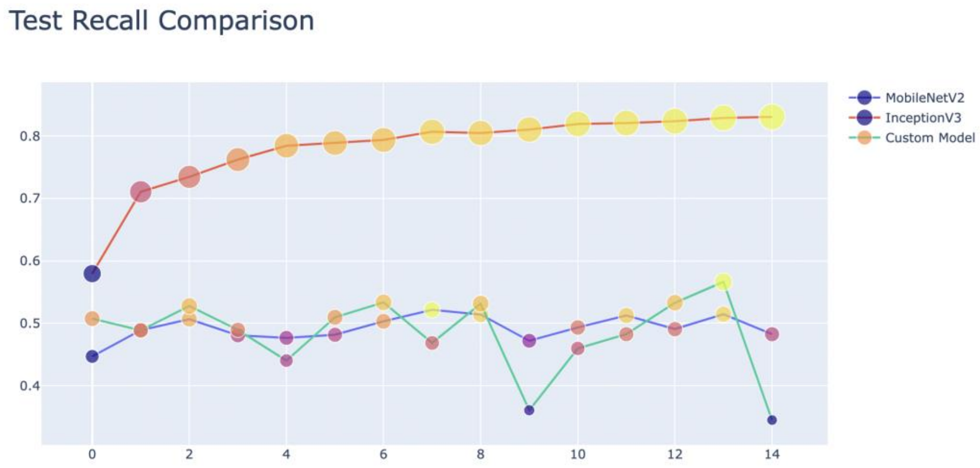
After individual implementation for each model we have compared the Train loss, Test loss, Test Accuracy, Test precision and Test recall for all three models. Below image shows the implementation of comparison for accuracy as this is the important factor in this research.

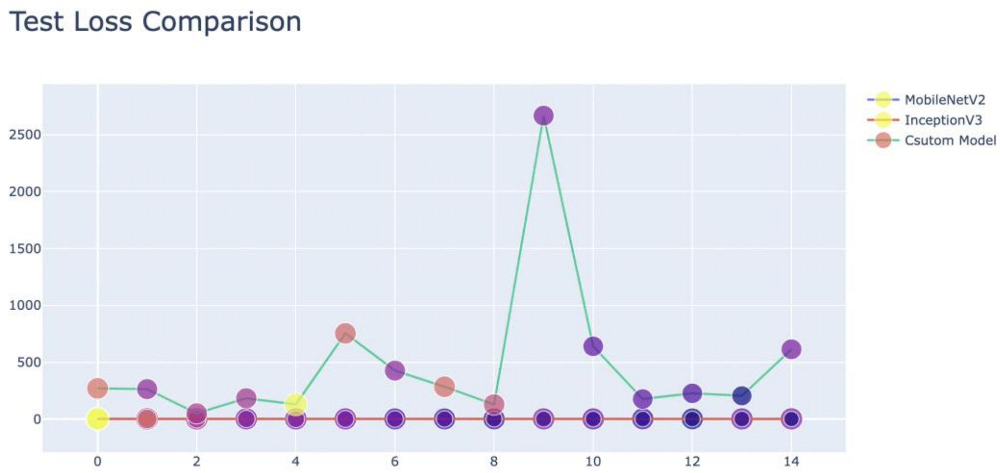
```
In [83]:
fig = go.Figure(data=go.Scatter(
    y=history_MobileNetV2_noAug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_MobileNetV2_noAug.history["val_accuracy"])*25,
                color=np.array(history_MobileNetV2_noAug.history["val_accuracy"])*25),
    name="MobileNetV2"
))


fig.add_trace(go.Scatter(
    y=history_InceptionV3_Aug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_InceptionV3_Aug.history["val_accuracy"])*25,
                color=np.array(history_InceptionV3_Aug.history["val_accuracy"])*25),
    name="InceptionV3"
))

fig.add_trace(go.Scatter(
    y=history_model_custom_Aug.history["val_accuracy"],
    mode='lines+markers',
    marker=dict(size=np.array(history_model_custom_Aug.history["val_accuracy"])*25,
                color=np.array(history_model_custom_Aug.history["val_accuracy"])*25),
    name="Custom Model"
))

fig.update_layout(title=go.layout.Title(text="Test Accuracy Comparison",
                                        font=go.layout.title.Font(size=25)))

fig.show()
```

Below the output graph for each comparison for all three models.









7 Model Selection

After implementing the models for 11 classes and comparing accuracy for each model we have chosen two models having higher accuracy which is Incepetionv3 and mobileNetV2 for 3 classes. After looking at input data we have chosen three classes Meat, dessert and soup having more number input images (sample images) which more important for good analysis.

1 Model Selection

```
1. Inceptionv3 (googleNet)
    |=> With Data Augmentation

2. MobileNetV2
    |=> With Data Augmentation
```

In [6]:

```
# Data with  preprocessing
img_gen = tf.keras.preprocessing.image.ImageDataGenerator(rotation_range = 40,
                                                          horizontal_flip=True,
                                                          shear_range = 0.2,
                                                          zoom_range = 0.2,
                                                          rescale=1./255.0,)

# we will not do data augmentation with validation data
# As we want to make model perform well in real life
val_img_gen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1/255)


train_data = img_gen.flow_from_directory(directory=train_dir,
                            target_size=target_size[0:2],
                            batch_size=4,
                           )
val_data  = val_img_gen.flow_from_directory(directory=val_dir,
                            target_size=target_size[0:2],
                            batch_size=1,
                           )
```

```
Found 4325 images belonging to 3 classes.
Found 1449 images belonging to 3 classes.
```

7.1 Inceptionv3

As we have seen our models are performing well with data augmentation so we have taken the augmented data for further analysis. The incpetionv3 implementation has shown in below.

1.1 Inceptionv3

In [8]:

```
# Defining model
# We will be training from scratch
model = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet', input_shape=target_size)

model_InceptionV3 = tf.keras.models.Sequential()
model_InceptionV3.add(model)
model_InceptionV3.add(tf.keras.layers.Flatten())
model_InceptionV3.add(tf.keras.layers.BatchNormalization())
model_InceptionV3.add(tf.keras.layers.Dense(256, activation='relu'))
model_InceptionV3.add(tf.keras.layers.Dropout(0.5))
model_InceptionV3.add(tf.keras.layers.BatchNormalization())
model_InceptionV3.add(tf.keras.layers.Dense(128, activation='relu'))
model_InceptionV3.add(tf.keras.layers.Dropout(0.5))
model_InceptionV3.add(tf.keras.layers.BatchNormalization())
model_InceptionV3.add(tf.keras.layers.Dense(len(classes_), activation='softmax'))

model_InceptionV3.layers[0].trainable=False
```

7.2 MobileNetv2

As we have seen our models are performing well with data augmentation so we have taken the augmented data for further analysis. The incpetionv3 implementation has shown in below.

1.2 MobileNetV2

In [10]:

```
# Defining model
# We will be training from scratch
model = tf.keras.applications.MobileNetV2(include_top=False,weights="imagenet", input_shape=target_size)

model_MobileNetV2 = tf.keras.models.Sequential()
model_MobileNetV2.add(model)
model_MobileNetV2.add(tf.keras.layers.Flatten())

model_MobileNetV2.add(tf.keras.layers.Dense(256, activation='relu'))
model_MobileNetV2.add(tf.keras.layers.Dropout(0.5))
model_MobileNetV2.add(tf.keras.layers.BatchNormalization())


model_MobileNetV2.add(tf.keras.layers.Dense(len(classes_), activation='softmax'))

model_MobileNetV2.layers[0].trainable=False

```

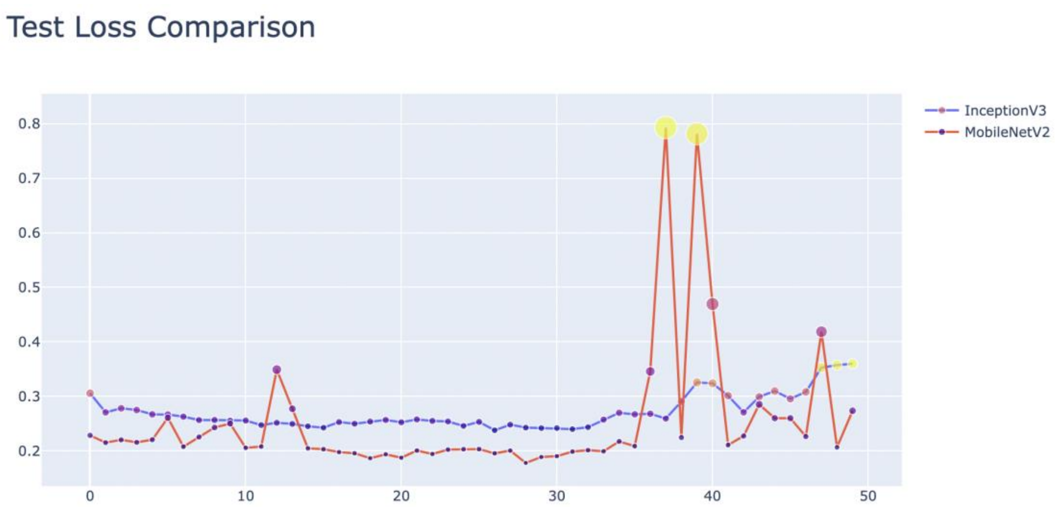
7.2 Model Comparison

After individual implementation for each model we have compared the Train loss, Test loss, Test Accuracy, Test precision and Test recall for both models. Below image shows the implementation of comparison for accuracy as this is the important factor in this research.

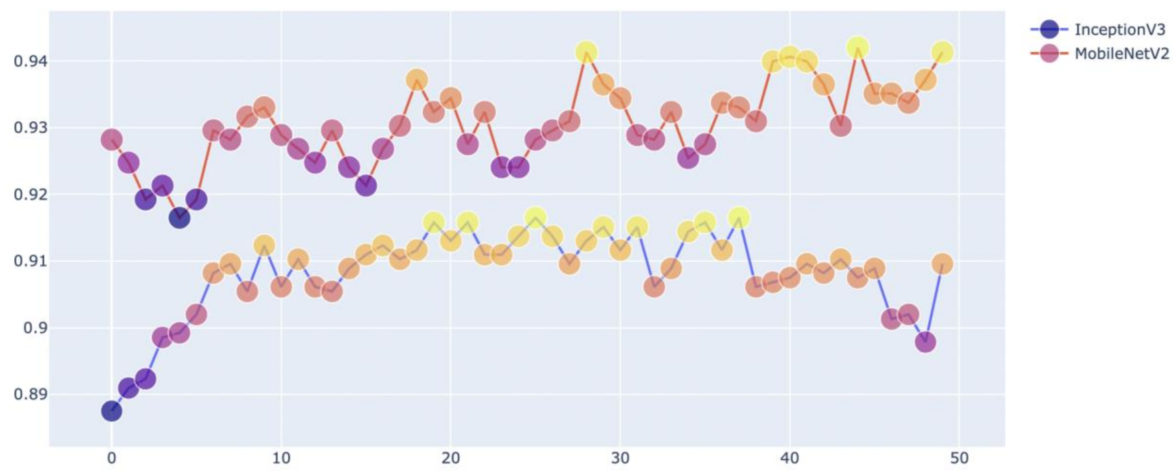
1.3 Model Comparison

In [19]:

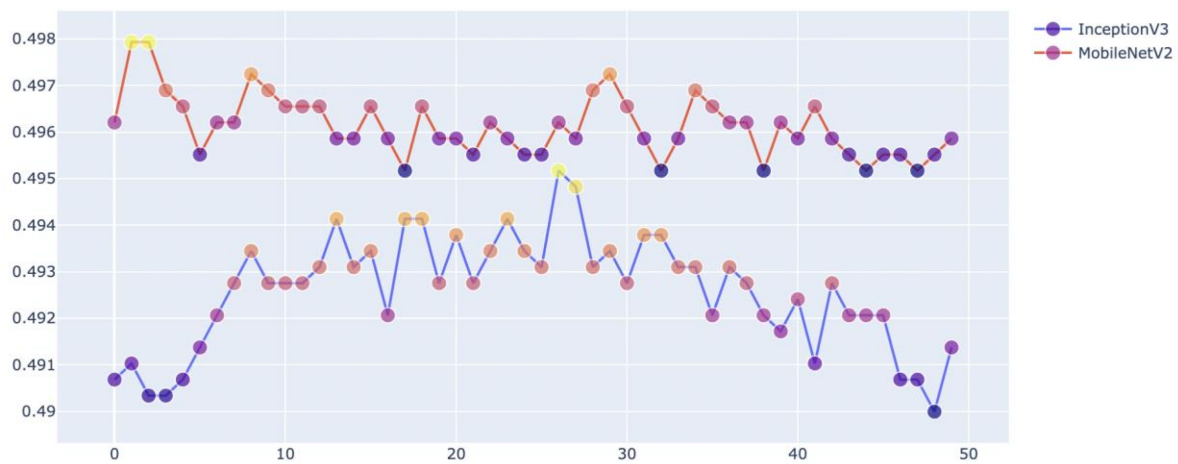
```
fig = go.Figure(data=go.Scatter(
    y=history_InceptionV3.history["val_loss"],
    mode='lines+markers',
    marker=dict(size=np.array(history_InceptionV3.history["val_loss"])*25,
                color=np.array(history_InceptionV3.history["val_loss"])*25),
    name="InceptionV3"
))


fig.add_trace(go.Scatter(
    y=history_MobileNetV2.history["val_loss"],
    mode='lines+markers',
    marker=dict(size=np.array(history_MobileNetV2.history["val_loss"])*25,
                color=np.array(history_MobileNetV2.history["val_loss"])*25),
    name="MobileNetV2"
))

fig.update_layout(title=go.layout.Title(text="Test Loss Comparison",
                                        font=go.layout.title.Font(size=25)))

fig.show()
```



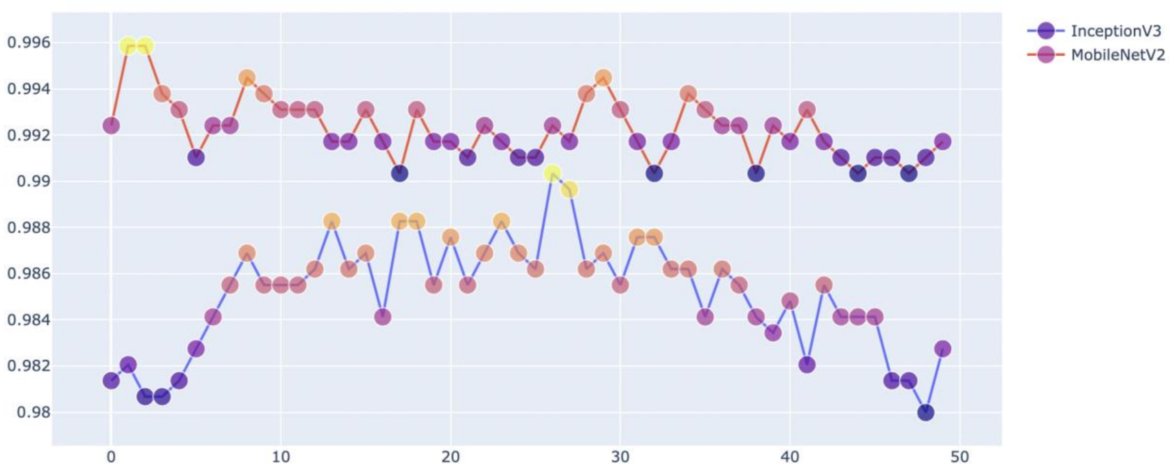
Test Accuracy Comparison



Test Precision Comparison



Test Recall Comparison



After looking at results for both the models we conclude that the MobileNetV2 is performing better than the Inceptionv2 giving accuracy of 94.15%. So we have saved the model as shown below.

1.4 Saving Final Model

In [30]:
```
model_MobileNetV2.save("MobileNetV2.h5")
```

7.3 Predicting Calories

We also added a one part in the analysis for predicting the calories for these three food items. The calories which we have defined are the standard calories rate for a particular food item it may vary if the quantity of the food changes. Implementation of the predication is shown below.

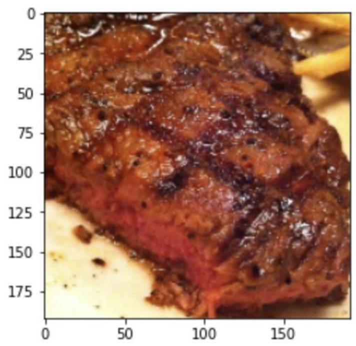
2 Predicting Calories

In [40]:
```
model = tf.keras.models.load_model("MobileNetV2.h5")
```

In [33]:
```
classes = ['Dessert', 'Meat', 'Soup']
calorie_values = {"Dessert":350, "Meat":143, "Soup":46}
```

In [51]:
```
img = plt.imread(r"Data_New\evaluation\Meat\1.jpg")
img = cv2.resize(img,target_size[0:2])
img = (img/255.0).reshape((1,192,192,3))

predicted_class_idx = np.argmax(model.predict([img]))
predicted_class = classes[predicted_class_idx]
calorie_intake = calorie_values[predicted_class]

print("Predicted Class => {}\
        Calorie Intake  => {}".format(predicted_class,calorie_intake))
plt.imshow(img[0])
plt.show()
```

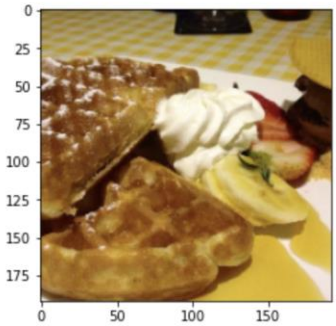
```
Predicted Class => Meat        Calorie Intake  => 143
```



In [52]:

```
img = plt.imread(r"Data_New\evaluation\Dessert\1.jpg")
img = cv2.resize(img,target_size[0:2])
img = (img/255.0).reshape((1,192,192,3))

predicted_class_idx = np.argmax(model.predict([img]))
predicted_class = classes[predicted_class_idx]
calorie_intake = calorie_values[predicted_class]

print("Predicted Class => {}\
       Calorie Intake  => {}".format(predicted_class,calorie_intake))
plt.imshow(img[0])
plt.show()
```

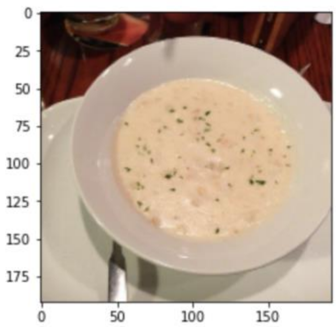
```
Predicted Class => Dessert       Calorie Intake  => 350
```



In [53]:

```
img = plt.imread(r"Data_New\evaluation\Soup\1.jpg")
img = cv2.resize(img,target_size[0:2])
img = (img/255.0).reshape((1,192,192,3))

predicted_class_idx = np.argmax(model.predict([img]))
predicted_class = classes[predicted_class_idx]
calorie_intake = calorie_values[predicted_class]

print("Predicted Class => {}\
       Calorie Intake  => {}".format(predicted_class,calorie_intake))
plt.imshow(img[0])
plt.show()
```

```
Predicted Class => Soup       Calorie Intake  => 46
```



8 Conclusion

This documentation covers all of the prerequisites, including hardware and software configuration, as well as the libraries and packages needed to build models. This report presents the entire project development process in a logical, succinct, and precise manner, making it easier to comprehend the implementation flow. So, we conclude that the MobileNetV2 model is performing better than the Inceptionv2.