Interim Report: Food Identification using Deep Learning

Project Overview

In this interim report, we present our progress in developing a deep learning model for food identification using the Food101 dataset. The objective of this project is to design and train a Convolutional Neural Network (CNN) model that can classify different types of food items based on images.

Milestone 1: Data Preprocessing and Basic CNN Model

Task 1: Import the Data

We uploaded the data on the google drive and imported from the shared colab notebook

```
my_folder_path = '/content/drive/MyDrive/Capstone Project/Food_101_10_50'
 my_folder_path_orig = '/content/drive/MyDrive/Capstone Project/Food_101'  #The original dataset
```

We successfully loaded the Food101 dataset, which contains 16,256 images of 17 classes of foods and also loaded a sample from the population with 500 images and 10 classes

Task 2: Map Training and Testing Images to Classes

We organized the dataset into training and testing sets for each food class. This enables us to train our model on a subset of the data and evaluate its performance on unseen images.

The Below Function splits the all folders within the given folder into train and test as per the desired ratio passed between 0-1 default is 0.8 ie 80% train and 20% test

```
def train_test_split(all_image_folder:str,train_data:float =0.8):

    import os,random,shutil

    folder_path = all_image_folder


    train_folder_path =  os.path.join(folder_path ,'train')
    if  os.path.exists(train_folder_path):
        shutil.rmtree(train_folder_path)
        os.mkdir(train_folder_path)
    else:

        os.mkdir(train_folder_path)
```

```
    test_folder_path =  os.path.join(folder_path ,'test')
    if  os.path.exists(test_folder_path):
        shutil.rmtree(test_folder_path)
        os.mkdir(test_folder_path)
    else:
        os.mkdir(test_folder_path)

    all_folder = os.listdir(folder_path)


    train = train_data

    for folder in all_folder:
        if folder == 'train' or folder == 'test':
          continue
        img_folder_path = os.path.join(folder_path,folder)
        if not os.path.exists(os.path.join(train_folder_path,folder)):
            os.mkdir(os.path.join(train_folder_path,folder))
        if not os.path.exists(os.path.join(test_folder_path,folder)):
            os.mkdir(os.path.join(test_folder_path,folder))

        total_images = len(os.listdir(img_folder_path))

        all_images = os.listdir(img_folder_path)

        train_images = all_images[:int(total_images*train)]
        test_images = all_images[int(total_images*train):]

        for images in train_images:
            shutil.copy(os.path.join(img_folder_path,images),os.path.jo
in(train_folder_path,folder))


        for images in test_images:
            shutil.copy(os.path.join(img_folder_path,images),os.path.jo
in(test_folder_path,folder))
```

The Below function takes the folder path and train/test kind of name we want on our csv file and converts the image folder name into classes and their respective image to a numpy array and returns a Pandas Dataframe and a h5 file

```
def get_dataframe(folder_path:str,df_type:str):
  import pandas as pd
  import os,cv2
  import h5py
  import numpy as np
```

```
  final_array = []
  class_array = []
  RESIZE_SHAPE = (254,254)
  df_images = pd.DataFrame()
  df_final = pd.DataFrame()
  all_classes = os.listdir(folder_path)
  for cls in all_classes:
    if '.' in cls:
      continue
    class_path = os.path.join(folder_path,cls)
    all_images = os.listdir(class_path)
    for image in all_images:
      image_path = os.path.join(class_path,image)
      img = cv2.imread(image_path)
      img = cv2.resize(img, RESIZE_SHAPE)
      final_array.append(img)
      class_array.append(cls)
      df_temp = pd.DataFrame(
          [
              {
                  'X':img,
                  'Y':cls,
                  'Image_name':image
              }
          ]
      )
      df_images = pd.concat([df_images,df_temp])
  final_array = np.array(final_array)
  class_array = np.array(class_array)
  with h5py.File(os.path.join(folder_path,df_type + ".h5"), "w") as hf:
    hf.create_dataset("images", data=final_array, compression="gzip", c
ompression_opts=9)
  df_images.reset_index(inplace=True,drop=True)
  df_images.to_csv(os.path.join(folder_path,df_type + '.csv'))
  return df_images
```

After mapping the train and test Data the shape of our data looks like this

```
Shape of x_test is (100, 254, 254, 3)
```

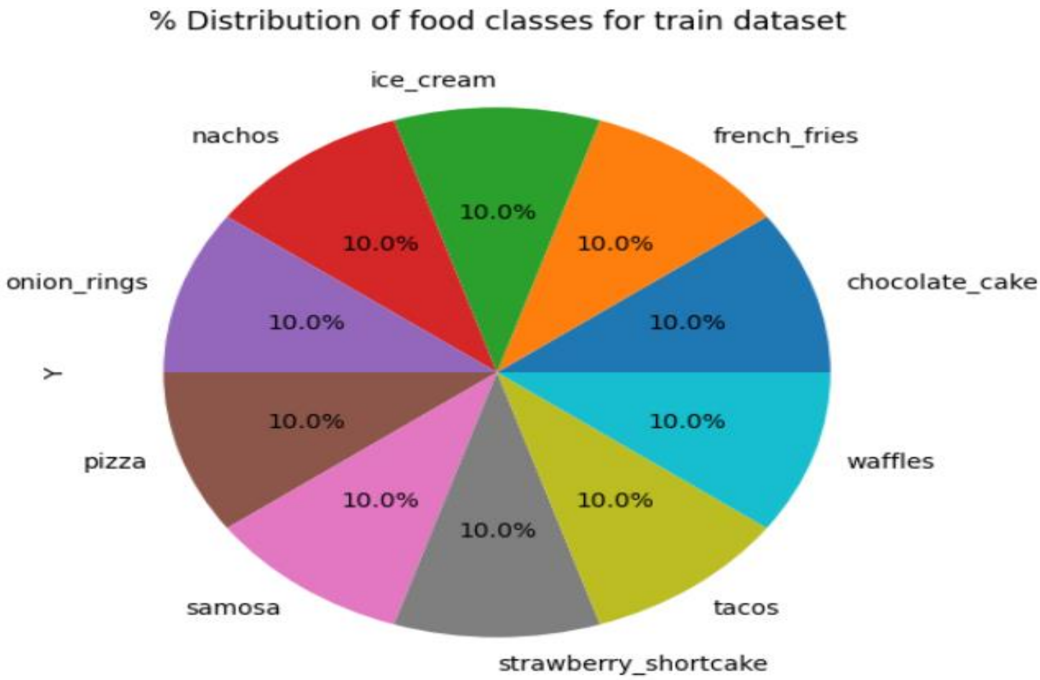
```
Shape of x_train is (400, 254, 254, 3)
```

We applied label encoding for the classes as the model takes numericals as an input

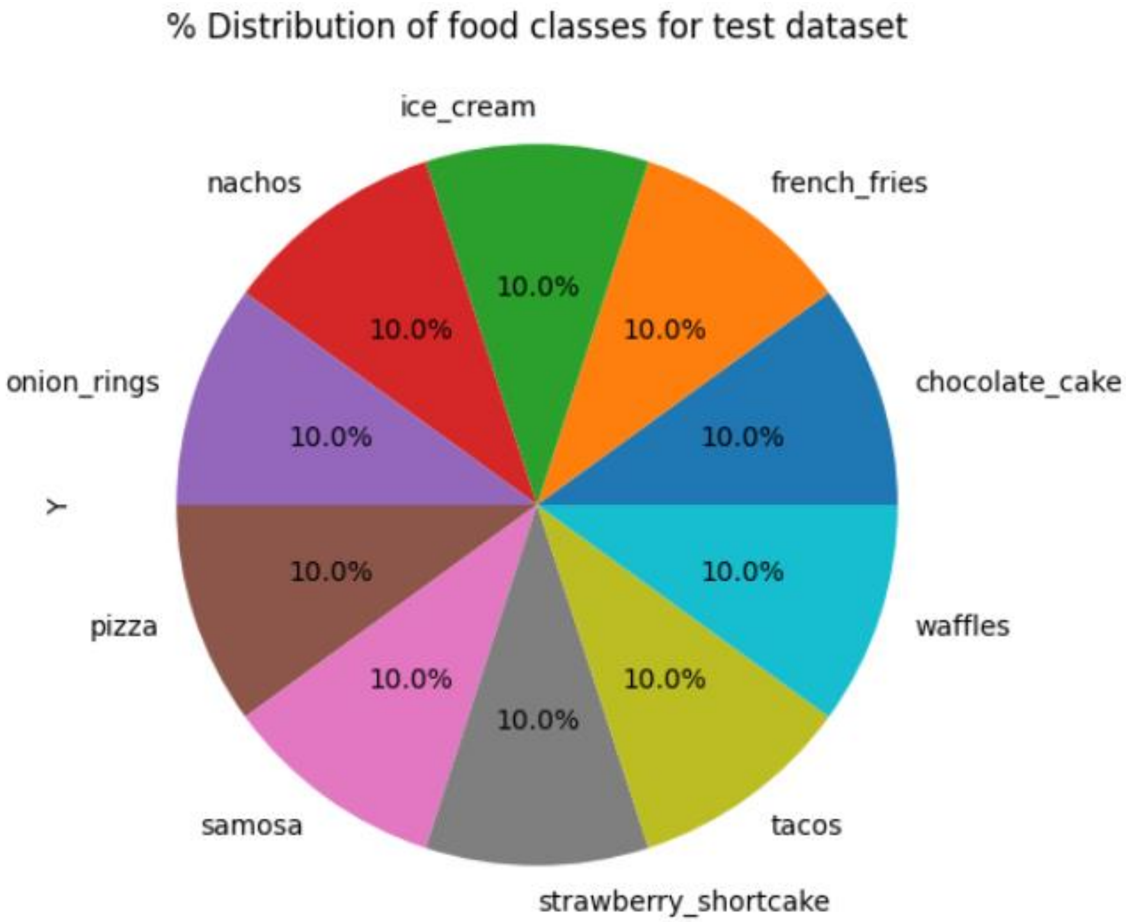
```
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from tensorflow.keras.utils import to_categorical

train_file_path = '/content/drive/MyDrive/Capstone Project/Food_101_10_50/train/train.csv'
test_file_path = '/content/drive/MyDrive/Capstone Project/Food_101_10_50/test/test.csv'

df_train = pd.read_csv(train_file_path)
df_test = pd.read_csv(test_file_path)

le = LabelEncoder()
le = le.fit(df_train['Y'])
y_train = le.transform(df_train['Y'])
y_test = le.transform(df_test['Y'])

num_classes = df_train['Y'].nunique()
y_train = to_categorical(y_train,num_classes=num_classes)
y_test = to_categorical(y_test,num_classes=num_classes)
```

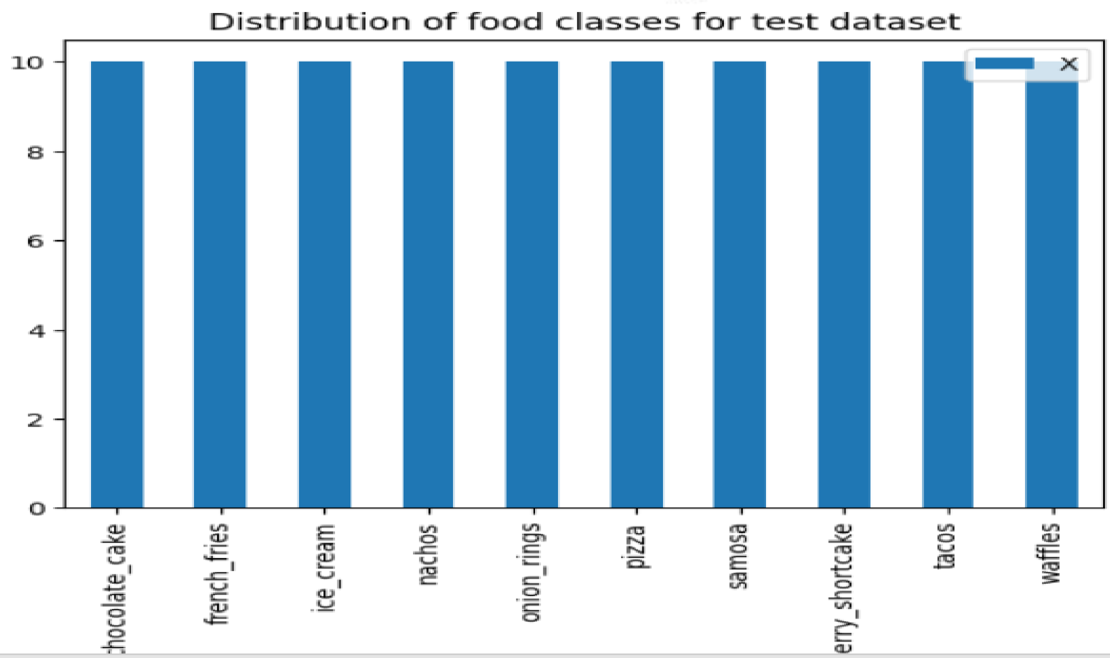
```
Shape of y_test is (100, 10)
```

```
Shape of y_train is (400, 10)
```


% Distribution of food classes for train dataset
ice_cream
french_fries
chocolate_cake
waffles
tacos
strawberry_shortcake
samosa
pizza
onion_rings
nachos
10.0%
Y


Distribution of food classes for training dataset
X
0
5
10
15
20
25
30
35
40
ocolate_cake
french_fries
ice_cream
nachos
onion_rings
pizza
samosa
ry_shortcake
tacos
waffles





We have balanced dataset with equally distributed classes 40 images each for 10 classes in training dataset and 10 images each for 10 classes in test dataset

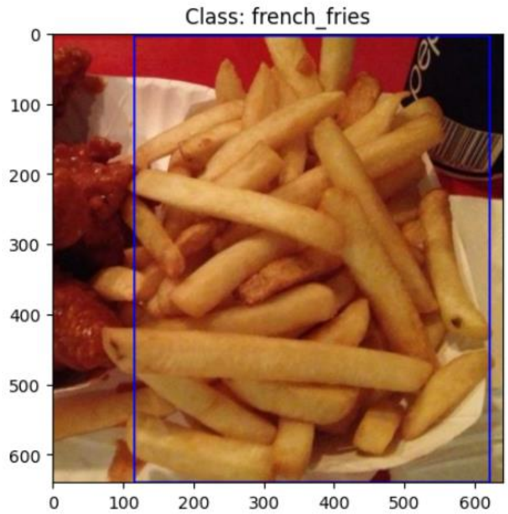
Task 3: Create Annotations for Training and Testing Images

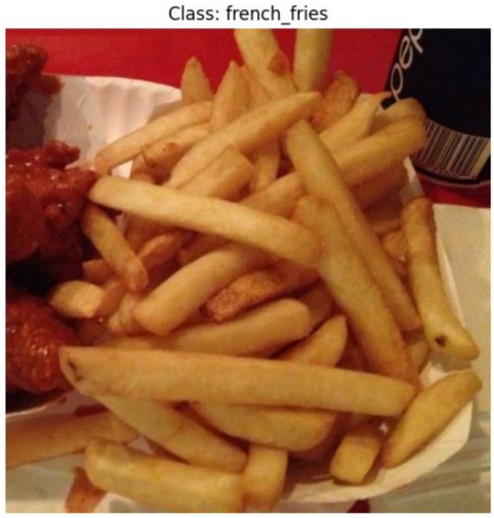
For the initial phase of our project, we manually annotated 10 food classes and selected 50 images from each class. Annotations include bounding box coordinates for each food item within the images.The annotations were done using the roboflow application

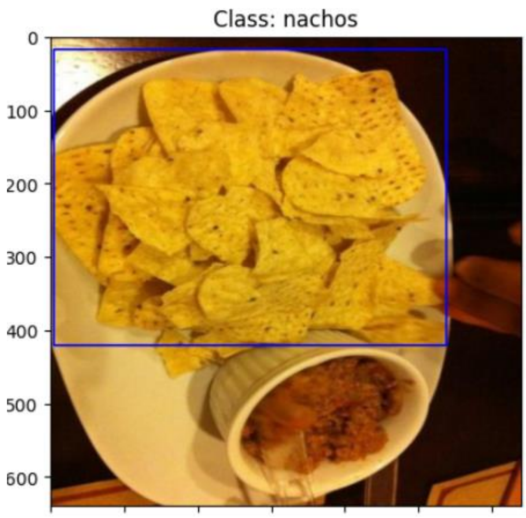
Roboflow was used as a tool to annotate images as it provides a smooth UI and also we can export the dataset to into specific data structure used by famous algorithm like Yolo,FasterRcnn which can further be used for transfer learning

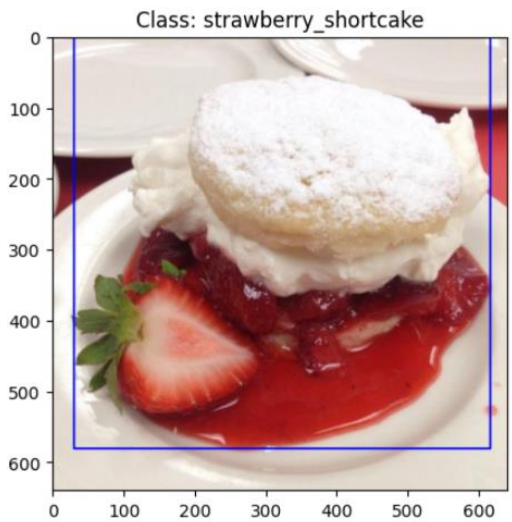
Task 4: Display Images with Bounding Boxes

We displayed the annotated images with bounding boxes to validate the accuracy of our annotations. This step allowed us to visually inspect the alignment between the annotations and the actual food items. Images with and without bounding box with their respective classes are displayed









Task 5: Design, Train, and Test Basic CNN Models

Summary:

We designed a basic CNN architecture for food item classification for both sample as well as the whole dataset. The model consists of convolutional layers followed by fully connected layers. Images were preprocessed by resizing and normalization before being fed into the model.We trained the CNN model using the annotated training images. After training, we evaluated the model's performance on the testing images. Preliminary results show that the model is overfit and has less accuracy.

Details:

1. Initially the model is trained and tested on a dataset with 10 classes and 50 images for each class.

   The pixel values of images that can range from 0-256 are normalized to between 0-1 for the train and test data. This is a common practice because the computation of high numeric values can be complex. This is achieved by dividing every pixel value by 255.

The **base model** layers are as below:

Input shape: 32 x 32 , 3 channel
Layer 1: Convolution (Number of Filters: 32,Kernel size: 3 x 3,Activation function: Relu)
Pooling: 2
Layer 2: Convolution(Number of Filters: 64,Kernel size: 3 x 3,Activation function: Relu)
Pooling: 2
Flatten layer
Layer 3: Neural network
Dense layer units: 64
Softmax output units: 10

Pooling layers downsample the feature maps and help to reduce overfitting
Flatten layer converts 2D arrays from pooled feature maps into a single long continuous linear vector to be fed into fully connected layer.
ReLU is used in hidden layer to avoid vanishing gradient problem and better computation performance , and Softmax function is used in last output layer to predict the outcome probabilities.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D, Dropout, Dense, Flatten, BatchNormalization, MaxPooling2D

# model architecture building
model = Sequential()

model.add(Convolution2D(filters = 32, kernel_size = 3, activation ='relu', input_shape = (254, 254, 3)))
model.add(MaxPooling2D(pool_size = 2))


model.add(Convolution2D(filters = 64, kernel_size = 3, activation = 'relu'))
model.add(MaxPooling2D(pool_size = 2))

model.add(Flatten())

# fully connected layer
model.add(Dense(units = 64,activation = 'relu'))

# Classification layer
model.add(Dense(units = 10, activation = 'softmax'))
```

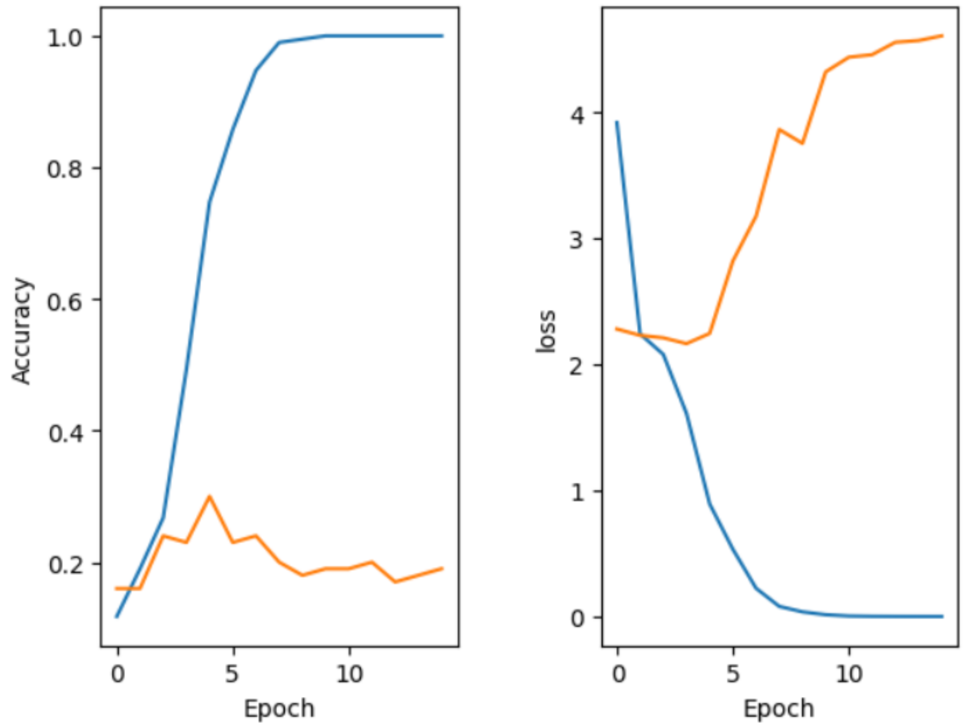
Model Summary:

```python
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 252, 252, 32)      896

 max_pooling2d_14 (MaxPoolin  (None, 126, 126, 32)     0
 g2D)

 conv2d_15 (Conv2D)          (None, 124, 124, 64)      18496

 max_pooling2d_15 (MaxPoolin  (None, 62, 62, 64)       0
 g2D)

 flatten_6 (Flatten)         (None, 246016)            0

 dense_12 (Dense)            (None, 64)                15745088

 dense_13 (Dense)            (None, 10)                650

=================================================================
Total params: 15,765,130
Trainable params: 15,765,130
Non-trainable params: 0
_________________________________________________________________
```

Accuracy and Loss after training the model is as below. The model is overfit and the accuracy is very low. The test score is 4.61 and test accuracy is 20%.

```
plot_accuracy(history)
```



Confusion matrix - printed after inverse_transform on the encoded y_test and y_pred

```
cm_df
```

	chocolate_cake	french_fries	ice_cream	nachos	onion_rings	pizza	samosa	strawberry_shortcake	tacos	waffles
chocolate_cake	6	0	2	0	1	0	0	0	0	1
french_fries	0	0	2	1	3	0	3	0	1	0
ice_cream	1	0	3	2	1	0	1	1	0	1
nachos	1	1	0	3	2	0	1	0	1	1
onion_rings	0	1	0	1	4	0	1	0	1	2
pizza	1	0	0	3	2	1	0	2	0	1
samosa	0	2	1	1	3	1	0	1	0	1
strawberry_shortcake	0	2	2	0	0	1	3	2	0	0
tacos	1	2	0	2	0	0	2	0	1	2
waffles	2	3	0	2	1	0	0	0	1	1

The class chocolate_cake has the highest True positives of 6 . It has 4 false negatives.
The class onion rings has 4 TP and 6 FN
The class ice cream and nachos have 3 TPs each and 7 FN each.
The remaining classes have very poor performance with 2,1 or 0 TP.

```python
from sklearn.metrics import classification_report
print(classification_report(y_test_orig, y_pred_orig, target_names=np.unique(df_train['Y'
```

```
                     precision    recall  f1-score   support

     chocolate_cake       0.50      0.60      0.55        10
       french_fries       0.00      0.00      0.00        10
          ice_cream       0.30      0.30      0.30        10
             nachos       0.20      0.30      0.24        10
        onion_rings       0.24      0.40      0.30        10
              pizza       0.33      0.10      0.15        10
             samosa       0.00      0.00      0.00        10
strawberry_shortcake       0.33      0.20      0.25        10
              tacos       0.20      0.10      0.13        10
            waffles       0.10      0.10      0.10        10

           accuracy                           0.21       100
          macro avg       0.22      0.21      0.20       100
       weighted avg       0.22      0.21      0.20       100
```

Highest F1 score is for Chocolate_cake class

2. In the next step, the model is tuned by adding drop out layers to the convolution and fully connected layers and batch normalization for the input and convolution layers to try and reduce the overfitting.

Batch normalisation has a regularising effect since it adds noise to the inputs of every layer and helps to reduce overfitting. It is used to normalize the output of the previous layers so each layer can learn more independently.

Dropouts are another regularization technique that is used to prevent overfitting in the model. Dropouts are added to randomly switch off some neurons of the network. This enhances the learning of the model and ensures the neurons are not codependent. A good value for dropout is 0.5(50% of neurons are dropped). More could affect the learning of the model.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D, Dropout, Dense, Flatten, BatchNormalization, MaxPooling2D

# model architecture building
model_1 = Sequential()

model_1.add(BatchNormalization(input_shape = (254, 254, 3)))

model_1.add(Convolution2D(filters = 32, kernel_size = 3, activation ='relu'))
model_1.add(BatchNormalization())
model_1.add(MaxPooling2D(pool_size = 2))
model_1.add(Dropout(0.5))


model_1.add(Convolution2D(filters = 64, kernel_size = 3, activation = 'relu'))
model_1.add(BatchNormalization())
model_1.add(MaxPooling2D(pool_size = 2))
model_1.add(Dropout(0.5))

model_1.add(Flatten())

# fully connected layer
model_1.add(Dense(units = 64,activation = 'relu'))
model_1.add(Dropout(0.5))

# Classification layer
model_1.add(Dense(units = 10, activation = 'softmax'))
```

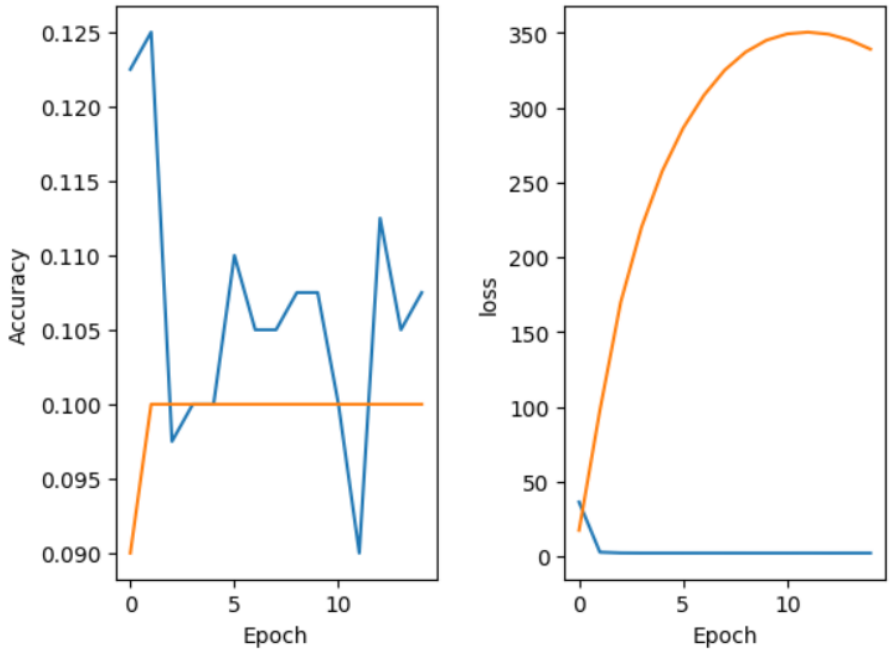
Model Summary:

```python
model_1.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 batch_normalization_7 (Batc  (None, 254, 254, 3)      12        
 hNormalization)                                                 
                                                                 
 conv2d_20 (Conv2D)          (None, 252, 252, 32)      896       
                                                                 
 max_pooling2d_20 (MaxPoolin  (None, 126, 126, 32)     0         
 g2D)                                                            
                                                                 
 dropout_16 (Dropout)        (None, 126, 126, 32)      0         
                                                                 
 conv2d_21 (Conv2D)          (None, 124, 124, 64)      18496     
                                                                 
 max_pooling2d_21 (MaxPoolin  (None, 62, 62, 64)       0         
 g2D)                                                            
                                                                 
 dropout_17 (Dropout)        (None, 62, 62, 64)        0         
                                                                 
 flatten_9 (Flatten)         (None, 246016)            0         
                                                                 
 dense_18 (Dense)            (None, 64)                15745088  
                                                                 
 dense_19 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 15,765,142
Trainable params: 15,765,136
Non-trainable params: 6
```

On training and testing the model, the accuracy remains 10% and the test score is 338.96. Accuracy and loss plots as below:

```
plot_accuracy(history)
```



The model is still overfit and the accuracy is low.

```
cm_df
```

```
4/4 [==============================] - 4s 786ms/step
```

	chocolate_cake	french_fries	ice_cream	nachos	onion_rings	pizza	samosa	strawberry_shortcake	tacos	waffles
chocolate_cake	0	0	0	0	0	0	0	0	10	0
french_fries	0	0	0	0	0	0	0	0	10	0
ice_cream	0	0	0	0	0	0	0	0	10	0
nachos	0	0	0	0	0	0	0	0	10	0
onion_rings	0	0	0	0	0	0	0	0	10	0
pizza	0	0	0	0	0	0	0	0	10	0
samosa	0	0	0	0	0	0	0	0	10	0
strawberry_shortcake	0	0	0	0	0	0	0	0	10	0
tacos	0	0	0	0	0	0	0	0	10	0
waffles	0	0	0	0	0	0	0	0	10	0

```
from sklearn.metrics import classification_report
print(classification_report(y_test_orig, y_pred_orig, target_names=np.unique(df_train['Y'])))
```

```
                      precision    recall  f1-score   support

      chocolate_cake       0.00      0.00      0.00        10
        french_fries       0.00      0.00      0.00        10
           ice_cream       0.00      0.00      0.00        10
              nachos       0.00      0.00      0.00        10
         onion_rings       0.00      0.00      0.00        10
               pizza       0.00      0.00      0.00        10
              samosa       0.00      0.00      0.00        10
strawberry_shortcake       0.00      0.00      0.00        10
               tacos       0.10      1.00      0.18        10
             waffles       0.00      0.00      0.00        10

            accuracy                           0.10       100
           macro avg       0.01      0.10      0.02       100
        weighted avg       0.01      0.10      0.02       100
```

All the images have been predicted as tacos here. Model is badly overfit.

3. In the next step, the model is further tuned by using techniques like data augmentation, L2 regularization, learning rate reduction and adjusting the batch size and epochs.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 252, 252, 32)      896       
                                                                 
 batch_normalization (BatchN  (None, 252, 252, 32)     128       
 ormalization)                                                   
                                                                 
 max_pooling2d (MaxPooling2D  (None, 126, 126, 32)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 124, 124, 64)      18496     
                                                                 
 batch_normalization_1 (Batc  (None, 124, 124, 64)     256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856     
                                                                 
 batch_normalization_2 (Batc  (None, 60, 60, 128)      512       
 hNormalization)                                                 
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 30, 30, 128)      0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 115200)            0         
                                                                 
 dense (Dense)               (None, 256)               29491456  
                                                                 
 batch_normalization_3 (Batc  (None, 256)              1024      
 hNormalization)                                                 
                                                                 
 dropout (Dropout)           (None, 256)               0         
                                                                 
 dense_1 (Dense)             (None, 128)               32896     
                                                                 
 batch_normalization_4 (Batc  (None, 128)              512       
 hNormalization)                                                 
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 29,621,322
Trainable params: 29,620,106
Non-trainable params: 1,216
```

```
# Define the number of classes in your dataset
num_classes = 10

# Train the model
batch_size = 32
epochs = 100

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

checkpoint_path = "/content/drive/MyDrive/Capstone
Project/Food_101_model.h5"

model_checkpoint = ModelCheckpoint(
    checkpoint_path,
    monitor='val_accuracy',
    save_best_only=True,
    mode='max',
    verbose=5)

# Define learning rate reduction based on plateau
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.1,  # Reduce learning rate by a factor of 0.1 when plateau
is detected
    patience=7,  # Number of epochs with no improvement after which
learning rate will be reduced
    min_lr=1e-6,  # Minimum learning rate
    verbose=1)




callbacks=[model_checkpoint, reduce_lr]

# Fit the model using data augmentation, model checkpoints, and
learning rate reduction
history = model.fit(
    datagen.flow(x_train, y_train, batch_size=batch_size),
    epochs=epochs,
    initial_epoch=0,
    validation_data=(x_test, y_test),
    callbacks=callbacks)
```

[9]

	chocolate_cake	french_fries	ice_cream	nachos	onion_rings	pizza	samosa	strawberry_shortcake	tacos	waffles
chocolate_cake	6	1	2	0	0	0	1	0	0	0
french_fries	0	3	0	1	2	0	0	0	4	0
ice_cream	0	0	6	1	0	0	0	0	3	0
nachos	0	1	1	2	1	0	1	0	4	0
onion_rings	0	3	0	1	3	0	0	0	3	0
pizza	0	0	0	3	1	0	1	0	4	1
samosa	0	2	2	1	1	0	2	0	2	0
strawberry_shortcake	0	0	2	1	0	1	1	0	5	0
tacos	0	1	0	0	0	0	1	0	8	0
waffles	0	1	2	1	0	0	1	0	5	0

```
from sklearn.metrics import classification_report
print(classification_report(y_test_orig, y_pred_orig, target_names=np.unique(df_train['Y'])))
```

```
                      precision    recall  f1-score   support

      chocolate_cake       1.00      0.60      0.75        10
        french_fries       0.25      0.30      0.27        10
           ice_cream       0.40      0.60      0.48        10
              nachos       0.18      0.20      0.19        10
         onion_rings       0.38      0.30      0.33        10
               pizza       0.00      0.00      0.00        10
              samosa       0.25      0.20      0.22        10
strawberry_shortcake       0.00      0.00      0.00        10
               tacos       0.21      0.80      0.33        10
             waffles       0.00      0.00      0.00        10
```

Data augmentation is a technique to create new training data from existing training data. This uses existing images to create transformed images by rotating, shifting, flipping, zooming the images etc. This technique is used here as we are using a small dataset with 50 images for each class.

An L2 regularization is used. It applies L2 regularization penalty on the convolution layers' kernel. This also helps with overfitting.

The learning rate is reduced on plateau by a factor of 0.1 after 7 epochs with no improvement. This helps to optimize the model.

Small batch size has a regularization effect so the batch size 32 is used. Epoch size is 100.

Model Checkpoints was used based on the best validation accuracy the model is saved which can further be trained or used to predict using the weights of the best model

Able to achieve improvement in accuracy and model was able to learn few classes better than other like chocolate cake and ice-cream have a better F1-score while pizza has the worst

Further by increasing the complexity of the model adding more convolutional layers training for more epochs and experimenting with the model structure also applying techniques like transfer learning we will be able to improve the performance of the model

Experiments on original Dataset with 17 classes and 17K images

Step-4 Base CNN model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D, Dropout, Dense,
Flatten, BatchNormalization, MaxPooling2D

# model architecture building
model = Sequential()

model.add(Convolution2D(filters = 32, kernel_size = 3, activation
='relu', input_shape = (254, 254, 3)))
model.add(MaxPooling2D(pool_size = 2))


model.add(Convolution2D(filters = 64, kernel_size = 3, activation =
'relu'))
model.add(MaxPooling2D(pool_size = 2))

model.add(Flatten())

# fully connected layer
model.add(Dense(units = 128,activation = 'relu'))
model.add(Dropout(0.5))

# Classification layer
model.add(Dense(units = 17, activation = 'softmax'))
```

.

```
32/102 [==============================] - 1s 12ms/step
```

	apple_pie	chocolate_cake	donuts	falafel	french_fries	hot_dog	ice_cream	nachos	onion_rings	pancakes	pizza	ravioli	samosa	spring_rolls	strawberry_shortcake	tacos	waffles
apple_pie	0	2	2	2	2	4	1	2	4	3	3	4	4	7	4	4	4
chocolate_cake	1	24	24	18	11	5	10	12	16	4	7	7	8	5	13	24	11
donuts	0	11	18	13	16	12	16	24	24	8	12	5	6	5	13	9	8
falafel	0	10	12	16	13	13	5	13	29	7	16	14	12	4	13	11	12
french_fries	0	3	3	9	25	10	6	19	32	9	22	9	11	4	16	13	9
hot_dog	0	6	9	15	22	12	8	13	31	5	15	7	14	7	18	8	10
ice_cream	1	16	10	16	16	9	15	12	30	7	8	12	9	5	16	11	7
nachos	0	5	9	6	15	6	2	19	56	7	16	7	9	5	15	12	11
onion_rings	0	3	5	9	30	10	4	18	24	7	14	12	7	11	25	12	9
pancakes	0	7	11	11	13	7	7	17	23	10	14	17	13	13	21	9	7
pizza	0	5	6	9	17	8	6	14	28	6	27	13	9	10	21	12	9
ravioli	1	5	6	10	12	14	6	11	24	15	18	27	13	12	13	9	4
samosa	0	3	7	7	15	10	6	15	29	7	15	12	22	10	16	16	10
spring_rolls	0	9	7	12	15	7	5	18	18	6	15	15	13	11	25	13	11
strawberry_shortcake	0	5	4	9	13	5	7	19	30	11	18	4	7	5	45	7	11
tacos	1	6	14	8	16	8	11	16	29	7	15	15	7	10	11	18	8
waffles	0	10	17	12	12	10	3	13	37	6	10	10	9	8	15	11	17

```
                        precision    recall  f1-score   support

            apple_pie       0.00      0.00      0.00        52
       chocolate_cake       0.18      0.12      0.15       200
               donuts       0.11      0.09      0.10       200
              falafel       0.09      0.08      0.08       200
         french_fries       0.10      0.12      0.11       200
              hot_dog       0.08      0.06      0.07       200
            ice_cream       0.13      0.07      0.09       200
               nachos       0.07      0.10      0.08       200
          onion_rings       0.05      0.12      0.07       200
             pancakes       0.08      0.05      0.06       200
                pizza       0.11      0.14      0.12       200
              ravioli       0.14      0.14      0.14       200
               samosa       0.13      0.11      0.12       200
         spring_rolls       0.08      0.06      0.07       200
strawberry_shortcake       0.15      0.23      0.18       200
                tacos       0.09      0.09      0.09       200
              waffles       0.11      0.09      0.09       200

             accuracy                           0.10      3252
            macro avg       0.10      0.10      0.10      3252
         weighted avg       0.10      0.10      0.10      3252
```

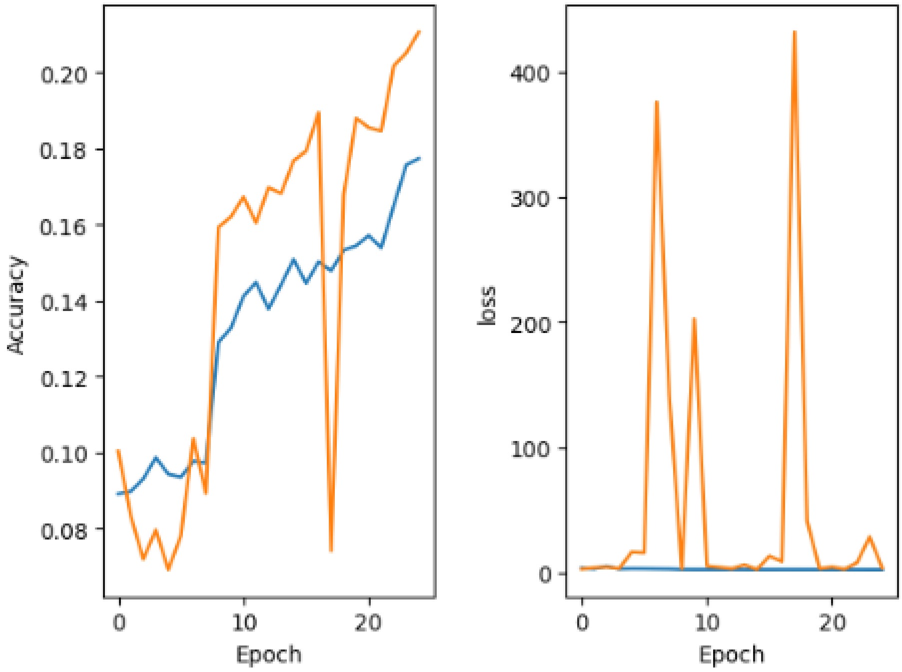
Step-5 Model tuned and modified on original dataset

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 252, 252, 32)      896

 batch_normalization (BatchN  (None, 252, 252, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 126, 126, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 124, 124, 64)      18496

 batch_normalization_1 (Batc  (None, 124, 124, 64)     256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 batch_normalization_2 (Batc  (None, 60, 60, 128)      512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 128)      0
 2D)

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 256)               29491456
```

```
 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 batch_normalization_4 (Batc  (None, 128)              512
 hNormalization)

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 17)                2193

=================================================================
Total params: 29,622,225
Trainable params: 29,621,009
Non-trainable params: 1,216
```

```
# Define the number of classes in your dataset
num_classes = 17

# Train the model
batch_size = 8
epochs = 25


checkpoint_path = "/content/drive/MyDrive/Capstone
Project/Food_101_orig_model.h5"

model_checkpoint = ModelCheckpoint(
    checkpoint_path,
    monitor='val_accuracy',
    save_best_only=True,
    mode='max',
    verbose=5)

# Define learning rate reduction based on plateau
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.1,  # Reduce learning rate by a factor of 0.2 when plateau
is detected
    patience=7,  # Number of epochs with no improvement after which
learning rate will be reduced
    min_lr=1e-6,  # Minimum learning rate
    verbose=1)
```

```
callbacks=[model_checkpoint, reduce_lr]

# Fit the model using data augmentation, model checkpoints, and
learning rate reduction
history = model.fit(
    x_train, y_train, batch_size=batch_size,
    epochs=epochs,
    initial_epoch=0,
    validation_data=(x_test, y_test),
    callbacks=callbacks)
```



	apple_pie	chocolate_cake	donuts	falafel	french_fries	hot_dog	ice_cream	nachos	onion_rings	pancakes	pizza	ravioli	samosa	spring_rolls	strawberry_shortcake	tacos	waffles
apple_pie	0	6	1	5	3	3	5	0	10	0	2	5	7	1	0	4	0
chocolate_cake	0	150	2	6	0	4	18	1	5	1	2	2	1	0	5	3	0
donuts	0	24	7	13	4	25	54	6	29	0	9	4	6	1	12	5	1
falafel	0	43	1	26	1	36	18	7	26	0	2	4	13	0	9	14	0
french_fries	1	9	1	2	48	15	11	11	72	0	3	6	2	1	1	17	0
hot_dog	0	5	2	8	17	55	19	14	28	0	9	3	11	0	1	28	0
ice_cream	1	33	9	10	1	16	75	11	8	0	3	4	4	4	11	10	0
nachos	0	7	3	4	21	14	12	45	44	0	13	4	4	0	5	24	0
onion_rings	0	13	0	6	24	11	4	8	110	0	5	1	10	0	1	7	0
pancakes	0	19	2	13	11	29	17	7	31	1	6	14	16	4	6	22	2
pizza	0	3	3	1	22	16	30	22	28	0	35	5	1	1	11	21	1
ravioli	1	9	2	7	19	15	7	10	45	0	10	31	13	5	5	20	1
samosa	0	10	3	11	21	20	7	10	59	0	6	10	27	1	2	13	0
spring_rolls	0	8	1	10	14	26	25	8	44	0	8	15	11	6	6	18	0
strawberry_shortcake	0	23	3	12	1	15	71	8	4	0	10	5	1	3	30	13	1
tacos	1	6	3	10	21	36	16	11	23	0	10	8	8	2	8	37	0
waffles	0	15	2	18	11	23	25	12	32	1	9	5	9	0	16	20	2

	precision	recall	f1-score	support
apple_pie	0.00	0.00	0.00	52
chocolate_cake	0.39	0.75	0.51	200
donuts	0.16	0.04	0.06	200
falafel	0.16	0.13	0.14	200
french_fries	0.20	0.24	0.22	200
hot_dog	0.15	0.28	0.20	200
ice_cream	0.18	0.38	0.24	200
nachos	0.24	0.23	0.23	200
onion_rings	0.18	0.55	0.28	200
pancakes	0.33	0.01	0.01	200
pizza	0.25	0.17	0.20	200
ravioli	0.25	0.15	0.19	200
samosa	0.19	0.14	0.16	200
spring_rolls	0.21	0.03	0.05	200
strawberry_shortcake	0.23	0.15	0.18	200
tacos	0.13	0.18	0.16	200
waffles	0.25	0.01	0.02	200
accuracy			0.21	3252
macro avg	0.21	0.20	0.17	3252
weighted avg	0.22	0.21	0.18	3252

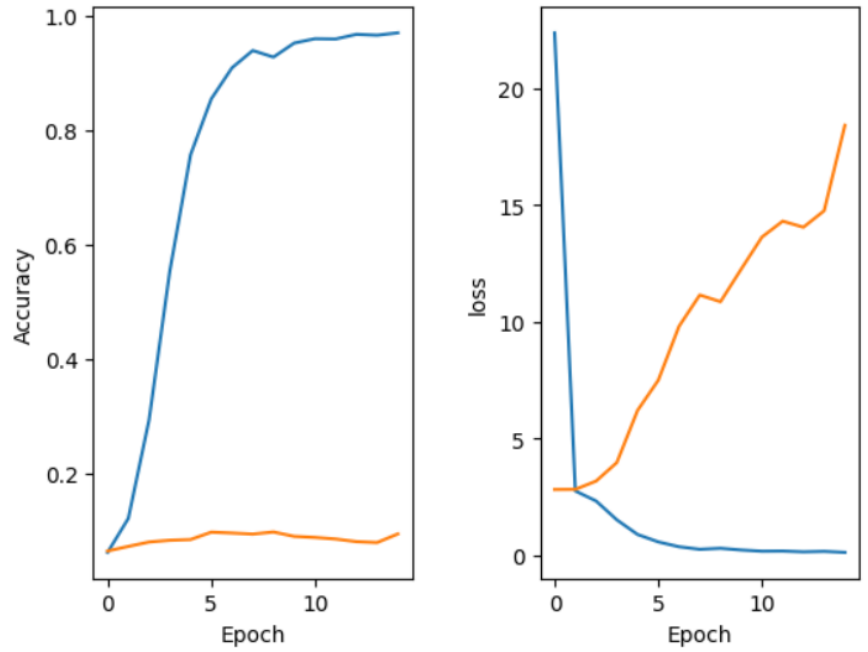
4. Then the original dataset with 17 classes was taken for training and the base model and tuned model were trained on this dataset.

The base model (step 1) is trained on original data set gives the below results:
The test score is 18.4 and accuracy is 9%.

```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 252, 252, 32)      896

 max_pooling2d_14 (MaxPoolin  (None, 126, 126, 32)     0
 g2D)

 conv2d_15 (Conv2D)          (None, 124, 124, 64)      18496

 max_pooling2d_15 (MaxPoolin  (None, 62, 62, 64)       0
 g2D)

 flatten_6 (Flatten)         (None, 246016)            0

 dense_12 (Dense)            (None, 64)                15745088

 dense_13 (Dense)            (None, 10)                650

=================================================================
Total params: 15,765,130
Trainable params: 15,765,130
Non-trainable params: 0
_________________________________________________________________
```

```
plot_accuracy(history)
```



The model performance has worsened.

5. Training the tuned model (Step 3) on the original dataset

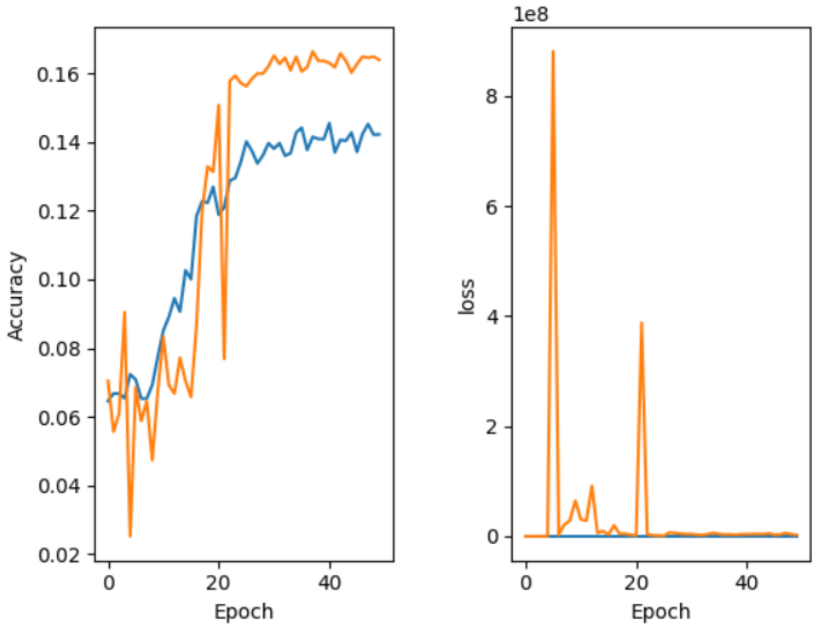
Model Summary:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 252, 252, 32)      896       
                                                                 
 batch_normalization (BatchN  (None, 252, 252, 32)     128       
 ormalization)                                                   
                                                                 
 max_pooling2d (MaxPooling2D  (None, 126, 126, 32)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 124, 124, 64)      18496     
                                                                 
 batch_normalization_1 (Batc  (None, 124, 124, 64)     256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856     
                                                                 
 batch_normalization_2 (Batc  (None, 60, 60, 128)      512       
 hNormalization)                                                 
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 30, 30, 128)      0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 115200)            0         
                                                                 
 dense (Dense)               (None, 256)               29491456  
                                                                 
 batch_normalization_3 (Batc  (None, 256)              1024      
 hNormalization)                                                 
                                                                 
 dropout (Dropout)           (None, 256)               0         
                                                                 
 dense_1 (Dense)             (None, 128)               32896     
                                                                 
 batch_normalization_4 (Batc  (None, 128)              512       
 hNormalization)                                                 
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 17)                2193      
                                                                 
=================================================================
Total params: 29,622,225
Trainable params: 29,621,009
Non-trainable params: 1,216
_________________________________________________________________
```

Accuracy and loss: Accuracy is 16 %.

```
plot_accuracy(history)
```



Overfitting is reduced but accuracy has not improved.

Challenges Faced

During the annotation process, we encountered challenges related to accurately annotating bounding boxes on irregularly shaped food items. We addressed this by adopting an iterative approach, refining our annotations based on visual inspection.

Next Steps

Moving forward, we plan to:

Experiment with different CNN architectures to improve classification accuracy.

Implement more data augmentation techniques to enhance model generalization.

Try and apply transfer learning on VGG-16 or Resnet-50 and unfreeze the final 2-3 layers and see if it benefits

Conclusion

The completion of the first milestone represents a significant step toward achieving our project objective. We have successfully laid the foundation for our food identification model by preprocessing data, annotating images, and training a basic CNN model. As we proceed, we aim to enhance the model's performance and explore advanced techniques in computer vision.