Nigerian food Classification

Group12\_Intro\_to\_AI

2024-08-05

# Nigerian Foods and Snacks Image Classifier

This project aims to classify images of Nigerian foods and snacks using the InceptionV3 architecture. The classifier provides predictions along with additional nutritional and geographical information about the food.

## Table of Contents

* [Overview](#overview)
* [Dataset](#dataset)
* [Setup](#setup)
* [Data Preprocessing](#data-preprocessing)
* [Model Training](#model-training)
* [Model Evaluation](#model-evaluation)
* [Confusion Matrix](#confusion-matrix)
* [Additional Information Integration](#additional-information-integration)
* [Deployment](#deployment)
* [Usage](#usage)
* [Contributing](#contributing)

## Overview

The project leverages the InceptionV3 model to classify Nigerian foods and snacks into various categories. The model is fine-tuned to provide high accuracy and includes additional features like data augmentation and class weight adjustments to handle imbalanced data.

## Dataset

The dataset used in this project can be downloaded from Kaggle. It contains images of various Nigerian foods and snacks divided into training, validation, and test sets.

## Setup

1. **Clone the repository:**

* git clone https://github.com/yourusername/nigerian-foods-classifier.git  
  cd nigerian-foods-classifier

1. **Install the required packages:**

* pip install -r requirements.txt

1. **Download the dataset:**

* Make sure you have a Kaggle account and have your Kaggle API key set up.
* !kaggle datasets download -d peaceedogun/nigerian-foods-and-snacks-multiclass

1. **Extract the dataset:**

* import zipfile  
  from pathlib import Path  
    
  zip\_file\_path = Path("C:/Users/user/OneDrive - Ashesi University/intro to ai/nigerian-foods-and-snacks-multiclass.zip")  
  extract\_dir = Path("C:/Users/user/OneDrive - Ashesi University/intro to ai/nigerian-foods-and-snacks")  
    
  extract\_dir.mkdir(parents=True, exist\_ok=True)  
    
  with zipfile.ZipFile(zip\_file\_path, 'r') as zip\_ref:  
   zip\_ref.extractall(extract\_dir)

1. **Set up paths to dataset:**

* train\_dir = r"C:\Users\user\OneDrive - Ashesi University\intro to ai\nigerian-foods-and-snacks\naija\_foods\content\naija\_foods\_multiclass\train"  
  test\_dir = r"C:\Users\user\OneDrive - Ashesi University\intro to ai\nigerian-foods-and-snacks\naija\_foods\content\naija\_foods\_multiclass\test"  
  val\_dir = r"C:\Users\user\OneDrive - Ashesi University\intro to ai\nigerian-foods-and-snacks\naija\_foods\content\naija\_foods\_multiclass\validation"  
  whole\_data = r"C:\Users\user\OneDrive - Ashesi University\intro to ai\nigerian-foods-and-snacks\naija\_foods\content\naija\_foods\_multiclass"

## Data Preprocessing

1. **Check and remove corrupted images:**

* from PIL import Image  
  import os  
    
  def check\_and\_remove\_corrupt\_images(directory):  
   corrupt\_images = []  
   for subdir, dirs, files in os.walk(directory):  
   for file in files:  
   filepath = os.path.join(subdir, file)  
   try:  
   with Image.open(filepath) as img:  
   img.verify()  
   except (IOError, SyntaxError, OSError) as e:  
   print(f'Removing bad file: {filepath}')  
   corrupt\_images.append(filepath)  
   os.remove(filepath)  
   return corrupt\_images  
    
  whole\_data = 'C:/Users/user/OneDrive - Ashesi University/intro to ai/nigerian-foods-and-snacks-multiclass'  
  corrupt\_images = check\_and\_remove\_corrupt\_images(whole\_data)  
  print(f'Found and removed {len(corrupt\_images)} corrupt images.')

1. **Load and preprocess data:**

* from tensorflow.keras.preprocessing.image import ImageDataGenerator  
    
  def load\_data(train\_fp, test\_fp, val\_fp):  
   datagen = ImageDataGenerator(  
   rescale=1./255,  
   shear\_range=0.2,  
   zoom\_range=0.2,  
   horizontal\_flip=True,  
   fill\_mode='nearest'  
   )  
   val\_datagen = ImageDataGenerator(rescale=1./255)  
   train\_gen = datagen.flow\_from\_directory(  
   train\_fp,  
   target\_size=(299, 299),  
   batch\_size=32,  
   class\_mode='categorical'  
   )  
   validation\_gen = val\_datagen.flow\_from\_directory(  
   val\_fp,  
   target\_size=(299, 299),  
   batch\_size=32,  
   class\_mode='categorical'  
   )  
   test\_gen = val\_datagen.flow\_from\_directory(  
   test\_fp,  
   target\_size=(299, 299),  
   batch\_size=32,  
   class\_mode='categorical'  
   )  
   return train\_gen, validation\_gen, test\_gen  
    
  train\_gen, validation\_gen, test\_gen = load\_data(train\_dir, test\_dir, val\_dir)  
  print(f"Number of training samples: {train\_gen.samples}")  
  print(f"Number of validation samples: {validation\_gen.samples}")

1. **Visualize class distribution:**

* import seaborn as sns  
  import matplotlib.pyplot as plt  
  import numpy as np  
    
  def plot\_class\_distribution(generator):  
   class\_counts = np.bincount(generator.classes)  
   class\_names = list(generator.class\_indices.keys())  
   sns.barplot(x=class\_names, y=class\_counts)  
   plt.xticks(rotation=90)  
   plt.title("Class Distribution")  
   plt.show()  
    
  plot\_class\_distribution(train\_gen)  
  plot\_class\_distribution(validation\_gen)  
  plot\_class\_distribution(test\_gen)

## Model Training

1. **Load and modify the InceptionV3 model:**

* from tensorflow.keras.applications import InceptionV3  
  from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout  
  from tensorflow.keras.models import Model  
  from tensorflow.keras.optimizers import Adam  
    
  base\_model = InceptionV3(weights='imagenet', include\_top=False, input\_shape=(299, 299, 3))  
  out = base\_model.output  
  pool = GlobalAveragePooling2D()(out)  
  pool = Dropout(0.5)(pool)  
  output = Dense(1024, activation='relu')(pool)  
  output = Dropout(0.5)(output)  
  predictions = Dense(train\_gen.num\_classes, activation='softmax')(output)  
  model = Model(inputs=base\_model.input, outputs=predictions)  
    
  for layer in base\_model.layers:  
   layer.trainable = False  
    
  model.compile(optimizer=Adam(learning\_rate=0.001), loss='categorical\_crossentropy', metrics=['accuracy'])

1. **Train the model:**

* from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping  
  from sklearn.utils.class\_weight import compute\_class\_weight  
    
  callbacks = [  
   ModelCheckpoint('best\_model.keras', save\_best\_only=True, monitor='val\_loss', mode='min'),  
   EarlyStopping(monitor='val\_loss', patience=5, mode='min', verbose=1)  
  ]  
    
  class\_weights = compute\_class\_weight(  
   class\_weight='balanced',  
   classes=np.unique(train\_gen.classes),  
   y=train\_gen.classes  
  )  
  class\_weights = dict(enumerate(class\_weights))  
    
  history = model.fit(  
   train\_gen,  
   steps\_per\_epoch=train\_gen.samples // train\_gen.batch\_size,  
   validation\_data=validation\_gen,  
   validation\_steps=validation\_gen.samples // validation\_gen.batch\_size,  
   epochs=10,  
   class\_weight=class\_weights,  
   callbacks=callbacks  
  )

## Model Evaluation

1. **Evaluate the model on the test set:**

* test\_loss, test\_acc = model.evaluate(test\_gen)  
  print(f'Test accuracy: {test\_acc}')

1. **Generate confusion matrix:**

* from sklearn.metrics import confusion\_matrix  
  import numpy as np  
    
  def get\_predictions\_and\_labels(generator, model):  
   all\_preds = []  
   all\_labels = []  
   for batch in generator:  
   imgs, labels = batch  
   preds = model.predict(imgs)  
   all\_preds.extend(np.argmax(preds, axis=1))  
   all\_labels.extend(np.argmax(labels, axis=1))  
   return np.array(all\_preds), np.array(all\_labels)  
    
  y\_pred, y\_true = get\_predictions\_and\_labels(test\_gen, model)  
  cm = confusion\_matrix(y\_true, y\_pred)  
  print('Confusion Matrix:\n', cm)

1. **Plot confusion matrix:**

* import seaborn as sns  
  import matplotlib.pyplot as plt  
    
  def plot\_confusion\_matrix(cm, class\_names):  
   plt.figure(figsize=(10, 7))  
   sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class\_names, yticklabels=class\_names)  
   plt.xlabel('Predicted Labels')  
   plt.ylabel('True Labels')  
   plt.title('Confusion Matrix')  
   plt.show()  
    
  class\_names = list(train\_gen.class\_indices.keys())  
  plot\_confusion\_matrix(cm, class\_names)

## Additional Information Integration

This section demonstrates how to integrate an additional dataset that includes detailed information about the foods, such as name, origin, nutritional content, health benefits, and more.

1. **Load additional information:**

* import pandas as pd  
    
  info\_df = pd.read\_excel('path\_to\_additional\_info.xlsx')  
  print(info\_df.head())

1. **Merge predictions with additional information:**

* def get\_food\_info(prediction\_index, info\_df):  
   food\_name = class\_names[prediction\_index]  
   food\_info = info\_df[info\_df['Name'] == food\_name]  
   return food\_info.to\_dict(orient='records')[0] if not food\_info.empty else {}  
    
  sample\_image, sample\_label = test\_gen[0][0][0], test\_gen[0][1][0]  
  sample\_prediction = model.predict(sample\_image[np.newaxis, ...])  
  predicted\_class = np.argmax(sample\_prediction)  
    
  food\_info = get\_food\_info(predicted\_class, info\_df)  
  print(food\_info)

## Deployment

Deploy the model on a user-friendly and interactive platform using Streamlit.

1. **Install Streamlit:**

* pip install streamlit

1. **Create a Streamlit app:**

* import streamlit as st  
  from tensorflow.keras.models import load\_model  
  from tensorflow.keras.preprocessing import image  
  import numpy as np  
  import pandas as pd  
    
  @st.cache(allow\_output\_mutation=True)  
  def load\_inception\_model():  
   model = load\_model('best\_model.keras')  
   return model  
    
  @st.cache  
  def load\_food\_info():  
   return pd.read\_excel('path\_to\_additional\_info.xlsx')  
    
  def get\_food\_info(prediction\_index, info\_df):  
   food\_name = class\_names[prediction\_index]  
   food\_info = info\_df[info\_df['Name'] == food\_name]  
   return food\_info.to\_dict(orient='records')[0] if not food\_info.empty else {}  
    
  model = load\_inception\_model()  
  info\_df = load\_food\_info()  
    
  st.title('Nigerian Foods and Snacks Classifier')  
  uploaded\_file = st.file\_uploader('Upload an image of the food', type=['jpg', 'png'])  
    
  if uploaded\_file is not None:  
   img = image.load\_img(uploaded\_file, target\_size=(299, 299))  
   st.image(img, caption='Uploaded Image', use\_column\_width=True)  
   img\_array = image.img\_to\_array(img)  
   img\_array = np.expand\_dims(img\_array, axis=0) / 255.0  
    
   prediction = model.predict(img\_array)  
   predicted\_class = np.argmax(prediction)  
    
   food\_info = get\_food\_info(predicted\_class, info\_df)  
    
   st.write('### Predicted Class:', class\_names[predicted\_class])  
   st.write('### Additional Information:')  
   for key, value in food\_info.items():  
   st.write(f'\*\*{key.capitalize()}\*\*: {value}')

1. **Run the Streamlit app:**

* streamlit run app.py

## Usage

1. **Fine-tune the best model:**

* best\_model = model  
    
  callbacks\_new = [  
   ModelCheckpoint('best\_newmodel.keras', save\_best\_only=True, monitor='val\_loss', mode='min'),  
   EarlyStopping(monitor='val\_loss', patience=5, mode='min', verbose=1)  
  ]  
    
  class\_weights\_new = compute\_class\_weight(  
   class\_weight='balanced',  
   classes=np.unique(train\_gen.classes),  
   y=train\_gen.classes  
  )  
  class\_weights\_new = dict(enumerate(class\_weights\_new))  
    
  history\_new = best\_model.fit(  
   train\_gen,  
   validation\_data=validation\_gen,  
   epochs=10,  
   class\_weight=class\_weights\_new,  
   callbacks=callbacks\_new  
  )  
    
  test\_loss, test\_acc = best\_model.evaluate(test\_gen)  
  print(f'Test accuracy: {test\_acc}')

## Contributing

If you would like to contribute to this project, please fork the repository and submit a pull request. Contributions are welcome!