dipcnn

November 26, 2024

[2]:
```python
import pandas as pd

# Load calorie data
calorie_data = pd.read_csv('/content/calorie_dataset.csv')
```

[3]:
```python
# Paths for training and testing datasets
train_dir = 'DataSets/food_photos'
test_dir = 'DataSets/test_photos'
```

[7]:
```python
import tarfile
import os

# Path to the tar file
tar_path = "/mnt/data/DataSets.tar"
extract_path = "/mnt/data/ExtractedDataSets"

# Extract the tar file
with tarfile.open('/content/DataSets.tar', "r") as tar:
    tar.extractall(path=extract_path)

print(f"Extracted files to: {extract_path}")
```

```
Extracted files to: /mnt/data/ExtractedDataSets
```

[8]:
```python
import tarfile

# Specify the tar file path
tar_file_path = '/mnt/data/DataSets.tar'

# Extract the tar file
with tarfile.open('/content/DataSets.tar', 'r') as tar:
    tar.extractall(path='/mnt/data/ExtractedDataSets')
```

[9]:
```python
import os

# Path to the directory containing the food categories (folders)
extracted_dir = '/mnt/data/ExtractedDataSets'
```

```
# Initialize empty lists for image paths and labels
image_paths = []
labels = []

# Loop through each folder (representing a food type)
for label in os.listdir(extracted_dir):
    label_dir = os.path.join(extracted_dir, label)

    # Check if it's a directory (ignore any non-directory files)
    if os.path.isdir(label_dir):
        # Loop through each image file in the folder
        for image_file in os.listdir(label_dir):
            # Get the full path to the image
            image_path = os.path.join(label_dir, image_file)

            # Append the image path and label
            image_paths.append(image_path)
            labels.append(label)

# Now, image_paths contains the full paths to the images, and labels contains␣
 ↪the corresponding food names
```

[10]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(image_paths, labels,␣
 ↪test_size=0.2, random_state=42)

# X_train, X_test contain the image paths for the training and testing sets
# y_train, y_test contain the corresponding labels
```

[12]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np

# Function to load and preprocess an image
def preprocess_image(image_path, target_size=(224, 224)):
    img = image.load_img('/content/pic_002 (1).jpg', target_size=target_size) ␣
 ↪# Resize image
    img_array = image.img_to_array(img) / 255.0  # Convert to array and␣
 ↪normalize (scale to [0, 1])
    return img_array

# Preprocess all images in the training and testing sets
X_train_preprocessed = np.array([preprocess_image('/content/pic_002 (1).jpg')␣
 ↪for img_path in X_train])
```

```
X_test_preprocessed = np.array([preprocess_image('/content/pic_002 (1).jpg')␣
 ↪for img_path in X_test])
```

[13]:
```
from sklearn.preprocessing import LabelEncoder

# Initialize the encoder
label_encoder = LabelEncoder()

# Fit and transform the labels (convert food names to integers)
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

[14]:
```
from tensorflow.keras import layers, models

# Build a simple CNN model
model = models.Sequential([
    layers.InputLayer(input_shape=(224, 224, 3)),  # Image size (224x224) and 3␣
 ↪color channels (RGB)
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(label_encoder.classes_), activation='softmax')  # Output␣
 ↪layer (one unit per class)
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
model.fit(X_train_preprocessed, y_train_encoded, epochs=10,␣
 ↪validation_data=(X_test_preprocessed, y_test_encoded))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/input_layer.py:26:
UserWarning: Argument `input_shape` is deprecated. Use `shape` instead.
  warnings.warn(

Epoch 1/10
1/1             2s 2s/step -
accuracy: 0.2500 - loss: 0.7387 - val_accuracy: 0.0000e+00 - val_loss: 19.1978
Epoch 2/10
1/1             1s 529ms/step -
accuracy: 0.7500 - loss: 4.7994 - val_accuracy: 0.0000e+00 - val_loss: 14.8157
Epoch 3/10
1/1             1s 513ms/step -
```

```
accuracy: 0.7500 - loss: 3.7039 - val_accuracy: 0.0000e+00 - val_loss: 7.4323
Epoch 4/10
1/1             1s 605ms/step -
accuracy: 0.7500 - loss: 1.8585 - val_accuracy: 0.0000e+00 - val_loss: 1.2794
Epoch 5/10
1/1             1s 512ms/step -
accuracy: 0.7500 - loss: 0.5644 - val_accuracy: 0.0000e+00 - val_loss: 0.7567
Epoch 6/10
1/1             1s 743ms/step -
accuracy: 0.7500 - loss: 0.6642 - val_accuracy: 0.0000e+00 - val_loss: 1.2230
Epoch 7/10
1/1             1s 819ms/step -
accuracy: 0.7500 - loss: 0.5672 - val_accuracy: 0.0000e+00 - val_loss: 1.8471
Epoch 8/10
1/1             1s 1s/step -
accuracy: 0.7500 - loss: 0.5905 - val_accuracy: 0.0000e+00 - val_loss: 1.8469
Epoch 9/10
1/1             1s 933ms/step -
accuracy: 0.7500 - loss: 0.5905 - val_accuracy: 0.0000e+00 - val_loss: 1.5305
Epoch 10/10
1/1             1s 623ms/step -
accuracy: 0.7500 - loss: 0.5655 - val_accuracy: 0.0000e+00 - val_loss: 1.0634
```

[14]:
```
<keras.src.callbacks.history.History at 0x7cb546cbe890>
```

[15]:
```
test_loss, test_accuracy = model.evaluate(X_test_preprocessed, y_test_encoded)
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
1/1             0s 66ms/step -
accuracy: 0.0000e+00 - loss: 1.0634
Test accuracy: 0.0000
```

[17]:
```
predictions = model.predict(X_test_preprocessed)

# Convert numeric predictions back to food names
predicted_labels = label_encoder.inverse_transform(np.argmax(predictions,␣
 ↪axis=1))
```

```
1/1             0s 309ms/step
```

[18]:
```
import matplotlib.pyplot as plt

# Get the number of samples in X_test_preprocessed
num_samples = len(X_test_preprocessed)

# Plot some random images and their predicted labels
# Ensure the loop iterates within the valid range of indices
for i in range(min(5, num_samples)):
```

```
    plt.imshow(X_test_preprocessed[i])
    plt.title(f"True: {y_test[i]} | Predicted: {predicted_labels[i]}")
    plt.axis('off')
    plt.show()
```

True: test_photos | Predicted: food_photos

[19]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
from tensorflow.keras.preprocessing import image
from sklearn.model_selection import train_test_split
```

[22]:
```
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load calorie dataset
calorie_data = pd.read_csv("calorie_dataset.csv")
```

[23]:
```
# Define directories
train_dir = "DataSets/food_photos"
```

```
test_dir = "DataSets/test_photos"

# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.2)
test_datagen = ImageDataGenerator(rescale=1.0 / 255)

# Load train and validation data
train_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode="categorical",
    subset="training"
)

val_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode="categorical",
    subset="validation"
)

# Load test data
test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode="categorical"
)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-23-c156dcea7f70> in <cell line: 10>()
      8
      9 # Load train and validation data
---> 10 train_data = train_datagen.flow_from_directory(
     11     train_dir,
     12     target_size=(150, 150),

/usr/local/lib/python3.10/dist-packages/keras/src/legacy/preprocessing/image.py
↪in flow_from_directory(self, directory, target_size, color_mode, classes,␣
↪class_mode, batch_size, shuffle, seed, save_to_dir, save_prefix, save_format,
↪follow_links, subset, interpolation, keep_aspect_ratio)
   1136         keep_aspect_ratio=False,
   1137     ):
```

```
-> 1138         return DirectoryIterator(
   1139             directory,
   1140             self,

/usr/local/lib/python3.10/dist-packages/keras/src/legacy/preprocessing/image.py
 ↪in __init__(self, directory, image_data_generator, target_size, color_mode,␣
 ↪classes, class_mode, batch_size, shuffle, seed, data_format, save_to_dir,␣
 ↪save_prefix, save_format, follow_links, subset, interpolation,␣
 ↪keep_aspect_ratio, dtype)
    451         if not classes:
    452             classes = []
--> 453             for subdir in sorted(os.listdir(directory)):
    454                 if os.path.isdir(os.path.join(directory, subdir)):
    455                     classes.append(subdir)

FileNotFoundError: [Errno 2] No such file or directory: 'DataSets/food_photos'
```

[24]:
```
import os

print(os.path.exists("DataSets/food_photos"))  # Should return True
print(os.path.exists("DataSets/test_photos"))  # Should return True
```

```
False
False
```

[25]:
```
train_dir = "/path/to/your/dataset/food_photos"
test_dir = "/path/to/your/dataset/test_photos"
```

[29]:
```
image_path = "/content/pic_002 (1).jpg"  # Represent the file path as a string
# Use the image_path variable to load your image data
```

[ ]:
```

```

[30]:
```
import os
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

[31]:
```
# Load calorie data
calorie_data = pd.read_csv("calorie_dataset.csv")

# Map food subcategories to calorie values
calorie_dict = calorie_data.set_index("FoodSubcategory")["Calories"].to_dict()
```

[33]:
```
import os

# Check main directory
print("DataSets exists:", os.path.exists("DataSets"))

# Check subdirectory
train_dir = "DataSets/food_photos"
print("Train directory exists:", os.path.exists(train_dir))

# List contents of the training directory
if os.path.exists(train_dir):
    print("Classes in train directory:", os.listdir(train_dir))
else:
    print("The train directory is missing or incorrectly specified.")
```

```
DataSets exists: True
Train directory exists: False
The train directory is missing or incorrectly specified.
```

[34]:
```
import os

print("Contents of 'DataSets':", os.listdir("DataSets"))
```

```
Contents of 'DataSets': []
```

[35]:
```
train_dir = "DataSets/<DataSets.tar>"
```

[41]:
```
# This is not Python code.  It appears to be a directory structure␣
 ↪representation.
# No changes are needed, but it cannot be executed as python code.
# To represent this in python you may want to create directories using shell␣
 ↪commands.
# Example:
# !mkdir -p DataSets/food_photos/Pizza

# The following lines represent the directory structure but are not executable␣
 ↪Python code.
# DataSets/
#     food_photos/
#         Pizza/
#             img1.jpg
#             img2.jpg
#             ...
#         Burger/
#             img1.jpg
#             img2.jpg
#             ...
```

```
#         Pasta/
#             img1.jpg
#             img2.jpg
#             ...
#         ...
#     test_photos/
#         test_img1.jpg
#         test_img2.jpg
#         ...
```

[43]:
```
import os

# Define the root directory for food images
train_dir = "/mnt/data/ExtractedDataSets"  # Changed to the extracted directory

# Check if the main directory exists
if os.path.exists(train_dir):
    print(f"Directory '{train_dir}' found.")

    # List subdirectories (food categories)
    subdirectories = os.listdir(train_dir)
    print("Subdirectories (food categories):", subdirectories)

    # Check if each subdirectory (food category) has images
    for food_category in subdirectories:
        category_path = os.path.join(train_dir, food_category)

        if os.path.isdir(category_path):
            print(f"Category '{food_category}' found. Checking images...")

            # List images in the category folder
            images = os.listdir(category_path)

            if len(images) > 0:
                print(f"  - Found {len(images)} images in '{food_category}'␣
 ↪category.")
            else:
                print(f"  - No images found in '{food_category}' category.")
        else:
            print(f"  - '{food_category}' is not a directory.")
else:
    print(f"Directory '{train_dir}' not found.")
```

```
Directory '/mnt/data/ExtractedDataSets' found.
Subdirectories (food categories): ['test_photos', 'food_photos']
Category 'test_photos' found. Checking images…
  - Found 3 images in 'test_photos' category.
```

```
Category 'food_photos' found. Checking images…
  - Found 3 images in 'food_photos' category.
```

[44]:
```python
import shutil
import os

# Define the source directory and the target directory
source_dir = "/mnt/data/ExtractedDataSets/food_photos"  # Your existing food_photos folder
target_dir = "/mnt/data/ExtractedDataSets/food_photos_organized"  # Directory to move organized images

# Create target subdirectories for each food category
categories = ['Pizza', 'Burger', 'Pasta']  # Example categories
for category in categories:
    category_path = os.path.join(target_dir, category)
    if not os.path.exists(category_path):
        os.makedirs(category_path)

# Example function to move files based on file names
def move_images_to_category(source_dir, target_dir, categories):
    for category in categories:
        category_path = os.path.join(target_dir, category)
        for img_name in os.listdir(source_dir):
            if category.lower() in img_name.lower():  # Check if the category name is in the image name
                source_image_path = os.path.join(source_dir, img_name)
                target_image_path = os.path.join(category_path, img_name)
                shutil.move(source_image_path, target_image_path)
                print(f"Moved {img_name} to {category} folder.")

move_images_to_category(source_dir, target_dir, categories)
```

```
Moved Pizza to Pizza folder.
Moved VegBurger to Burger folder.
```

[45]:
```python
train_dir = "/mnt/data/ExtractedDataSets/food_photos_organized"
test_dir = "/mnt/data/ExtractedDataSets/test_photos"
```

[47]:
```python
from tensorflow.keras.optimizers import Adam # Import the Adam optimizer

# ... (rest of your code) ...

# Compile the model
model.compile(optimizer=Adam(), loss="categorical_crossentropy", metrics=["accuracy"])
```

```
# ... (rest of your code) ...

# Set up data augmentation for training data
train_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.2)
test_datagen = ImageDataGenerator(rescale=1.0 / 255)

# Load the training and validation data
train_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode="categorical",
    subset="training"
)

val_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode="categorical",
    subset="validation"
)

# Build the CNN model (from previous example)
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(150, 150, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation="relu"),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation="relu"),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation="relu"),
    Dense(len(train_data.class_indices), activation="softmax")  # Number of␣
  ↪food categories
])

# Compile the model
model.compile(optimizer=Adam(), loss="categorical_crossentropy",␣
  ↪metrics=["accuracy"])

# Train the model
model.fit(train_data, epochs=10, validation_data=val_data)

# Save the trained model
model.save("food_recognition_model.h5")
```

```
Found 895 images belonging to 3 classes.
Found 222 images belonging to 3 classes.

/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/10

/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

28/28            51s 2s/step -
accuracy: 0.4948 - loss: 1.0495 - val_accuracy: 0.7252 - val_loss: 0.7315
Epoch 2/10
28/28            47s 2s/step -
accuracy: 0.7882 - loss: 0.4973 - val_accuracy: 0.8153 - val_loss: 0.3933
Epoch 3/10
28/28            49s 2s/step -
accuracy: 0.8799 - loss: 0.3205 - val_accuracy: 0.8333 - val_loss: 0.3906
Epoch 4/10
28/28            47s 2s/step -
accuracy: 0.8831 - loss: 0.2937 - val_accuracy: 0.8919 - val_loss: 0.2367
Epoch 5/10
28/28            47s 2s/step -
accuracy: 0.8998 - loss: 0.2532 - val_accuracy: 0.8694 - val_loss: 0.2963
Epoch 6/10
28/28            49s 2s/step -
accuracy: 0.9167 - loss: 0.1903 - val_accuracy: 0.9189 - val_loss: 0.1970
Epoch 7/10
28/28            82s 2s/step -
accuracy: 0.9196 - loss: 0.1875 - val_accuracy: 0.9189 - val_loss: 0.1831
Epoch 8/10
28/28            46s 2s/step -
accuracy: 0.9476 - loss: 0.1320 - val_accuracy: 0.9369 - val_loss: 0.1666
Epoch 9/10
28/28            46s 2s/step -
accuracy: 0.9578 - loss: 0.0914 - val_accuracy: 0.9685 - val_loss: 0.1275
Epoch 10/10
28/28            46s 2s/step -
accuracy: 0.9704 - loss: 0.0887 - val_accuracy: 0.9324 - val_loss: 0.1446
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

[50]:
```
import pandas as pd

# Load your calorie dataset
calorie_data = pd.read_csv('/content/calorie_dataset.csv')

# Inspect the first few rows to check the structure of the dataset
print(calorie_data.head())
```

```
           Food            FoodSubcategory  Quantity ServingSize(g)  Fat(g)  \
0        Burger              Cheese Burger         1          100 g   27.50
1        Burger                Beef Burger         1          100 g   10.09
2        Burger             Chicken Burger         1          100 g   14.81
3        Burger              Veggie Burger         1          100 g   12.48
4  French Fries  French Fries, Oven Heated         1          117 g    3.39

   Calories
0       297
1       264
2       286
3       261
4       133
```

[51]:
```
print(calorie_data.columns)
```

```
Index(['Food', 'FoodSubcategory', 'Quantity', 'ServingSize(g)', 'Fat(g)',
       'Calories'],
      dtype='object')
```

[52]:
```
calorie_dict = dict(zip(calorie_data['Food'], calorie_data['Calories']))
```

[54]:
```
# Create a dictionary from the dataset
calorie_dict = dict(zip(calorie_data['Food'], calorie_data['Calories']))

# Print the calorie dictionary to verify
print(calorie_dict)
```

```
{'Burger': 261, 'French Fries': 152, 'Apple': 17, 'Apple Pie': 106, 'Biriyani':
341, 'Pizza': 2248, 'Cake': 3956, 'Dosa': 392, 'Noodles': 910, 'Coffee': 62}
```

[56]:
```
import pandas as pd

# Load your calorie dataset
```

```
calorie_data = pd.read_csv('/content/calorie_dataset.csv')

# Inspect the first few rows to check the structure
print(calorie_data.head())

# Create a dictionary from the dataset
calorie_dict = dict(zip(calorie_data['Food'], calorie_data['Calories']))

# Print the calorie dictionary to verify
print(calorie_dict)

# Now you can use this dictionary for calorie estimation as shown earlier
food_item = 'Pizza'  # Example food item recognized by your model
calories = calorie_dict.get(food_item, "Calories data not available")
print(f"The estimated calories for {food_item} are: {calories}")
```

```
          Food            FoodSubcategory  Quantity ServingSize(g)  Fat(g)  \
0       Burger              Cheese Burger         1          100 g   27.50
1       Burger                Beef Burger         1          100 g   10.09
2       Burger             Chicken Burger         1          100 g   14.81
3       Burger              Veggie Burger         1          100 g   12.48
4  French Fries  French Fries, Oven Heated         1          117 g    3.39

   Calories
0       297
1       264
2       286
3       261
4       133
{'Burger': 261, 'French Fries': 152, 'Apple': 17, 'Apple Pie': 106, 'Biriyani':
341, 'Pizza': 2248, 'Cake': 3956, 'Dosa': 392, 'Noodles': 910, 'Coffee': 62}
The estimated calories for Pizza are: 2248
```

[58]:
```
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
import pandas as pd

# Load your calorie dataset and create the calorie_dict
calorie_data = pd.read_csv('/content/calorie_dataset.csv')
calorie_dict = dict(zip(calorie_data['Food'], calorie_data['Calories']))

# Print the calorie dictionary to verify
print(calorie_dict)

# Load your trained food recognition model (replace with your model's path)
```

```
model = load_model('food_recognition_model.h5')  # Ensure you have your model␣
 ↪path correct

# Function to load and preprocess the image for prediction
def prepare_image(img_path):
    img = image.load_img(img_path, target_size=(150, 150))  # Resize to match␣
 ↪model input size
    img_array = image.img_to_array(img)  # Convert image to array
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    img_array = img_array / 255.0  # Normalize pixel values to [0, 1]
    return img_array

# Function to predict food item and estimate calories
def recognize_food_and_calories(img_path, train_data): # Add train_data as an␣
 ↪argument
    img_array = prepare_image(img_path)  # Preprocess the image
    predictions = model.predict(img_array)  # Predict the class (food item)

    # Get the index of the class with the highest probability
    predicted_class_idx = np.argmax(predictions, axis=1)[0]

    # Get the class label (food name)
    # Access class_indices from the train_data ImageDataGenerator
    class_labels = list(train_data.class_indices.keys())
    predicted_food = class_labels[predicted_class_idx]

    # Lookup the predicted food in the calorie dictionary
    calories = calorie_dict.get(predicted_food, "Calories data not available")

    return predicted_food, calories

# Example usage
img_path = '/content/pic_002 (1).jpg'  # Replace with the actual image path

# Get food item and calorie estimate
# Pass train_data to recognize_food_and_calories
food_item, calorie_estimate = recognize_food_and_calories(img_path, train_data)

print(f"Food Item: {food_item}")
print(f"Estimated Calories: {calorie_estimate}")
```

```
{'Burger': 261, 'French Fries': 152, 'Apple': 17, 'Apple Pie': 106, 'Biriyani':
341, 'Pizza': 2248, 'Cake': 3956, 'Dosa': 392, 'Noodles': 910, 'Coffee': 62}
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
```

```
1/1             0s 420ms/step
Food Item: Pizza
Estimated Calories: 2248
```