A

Project Report

On

FoodSenseAI Identifying Food Categories using Deep Learning

Prepared by:

Vishw Mewada - 400488026

Nirav Makwana - 400488532

Om Tank - 400551740

Under the guidance of

Dr. Anwar Mirza

Submitted to

McMaster University

Master of Engineering

in Systems and Technology

SEP 740 – Deep Learning

Acknowledgement

We extend our heartfelt gratitude to the remarkable personalities who have been invaluable in supporting and guiding us throughout this project. Their unwavering support has played a pivotal role in ensuring the successful completion of this endeavor, and we are deeply thankful for their contributions.

First and foremost, we express our profound appreciation to our esteemed Professor, Dr. Anwar Mirza. Their dedication to our project and provision of essential resources from the college has been instrumental in shaping our research journey. We are truly grateful for their time, mentorship, and continuous encouragement, without which this project would never have come to fruition. Their guidance has been invaluable, and we are fortunate to have had them as our mentor.

This project has been enriched by the unwavering support and assistance from various other individuals, whose contributions have made a significant difference. To all those who have generously offered their expertise and encouragement, we offer our sincere thanks.

Completing this project would not have been possible without the support of these remarkable individuals, and we are honored to have worked with them. Their guidance and belief in our abilities have inspired us to push the boundaries and achieve our best. We express our heartfelt gratitude to everyone who has been a part of this journey, as this project's success is a testament to the power of collaboration and collective effort.

Once again, we extend our deepest appreciation to each person who has played a role in our project's development. Their support has been a driving force behind our accomplishments, and we are forever indebted to them for their kindness and dedication.

Abstract

This project aims to develop an advanced model capable of accurately classifying food images into 101 distinct categories. Utilizing the comprehensive Food-101 dataset, this project endeavors to train and validate a deep learning model that achieves high accuracy and efficiency.

Our methodology incorporates TensorFlow and TensorFlow Datasets to streamline data handling and preprocessing. We employ EfficientNetB0, a state-of-the-art convolutional neural network, for feature extraction, chosen for its balance of computational efficiency and high accuracy.

To enhance model performance and training speed, we implement mixed precision training, which leverages both 16-bit and 32-bit floating-point types to optimize memory usage and computational efficiency. This technique significantly accelerates training while maintaining model accuracy.

We utilize a variety of callbacks to fine-tune the training process:

- **ReduceLROnPlateau**: This callback is employed to reduce the learning rate when a plateau in model performance is detected, helping to avoid overfitting and improve convergence.
- **TensorBoard**: This callback is used for comprehensive monitoring of the training process, providing visual insights into model performance metrics and facilitating debugging and optimization.
- **ModelCheckpoint**: This callback saves the model at specified intervals, ensuring that the best-performing model is preserved during training, which is crucial for preventing the loss of valuable progress.
- **EarlyStopping**: This callback stops training when no improvement in model performance is detected over a set number of epochs, helping to save time and computational resources while preventing overfitting.

The project places a strong emphasis on meticulous data preparation, sophisticated model design, and thorough optimization of the training process. We document our experiments in detail, highlighting the various strategies employed, the outcomes achieved, and the challenges faced.

Our primary goal is to achieve superior classification accuracy, ensuring the model's reliability and practical applicability. By systematically addressing the challenges encountered during the development process and leveraging advanced techniques and tools, this project aims to make a significant contribution to the field of food image classification using cutting-edge deep learning methodologies.

Table of contents

1. Introduction

1.1. Project summary:

The project aims to classify food images into 101 predefined categories using the EfficientNetB0 model. The implementation of various callbacks, including EarlyStopping, ReduceLROnPlateau, and TensorBoard, was pivotal in optimizing the training process and preventing overfitting. By fine-tuning all layers of the EfficientNetB0 model, we maximized the potential of the pre-trained network, significantly enhancing accuracy and generalization.

This work underscores the potential of deep learning in handling complex image classification tasks, providing a robust foundation for future advancements. Future enhancements could involve expanding the dataset, further optimizing the model architecture, and deploying the model in real-world applications. The combination of a strong base model, thorough fine-tuning, and strategic use of callbacks showcases the effectiveness of deep learning techniques in achieving high classification accuracy and reliable performance.

1.2. Purpose:

This project's purpose is to develop a robust deep learning model capable of accurately classifying food images into 101 predefined categories. By leveraging the EfficientNetB0 model and implementing various optimization techniques, the project aims to push the boundaries of image classification accuracy and generalization. Additionally, the project seeks to demonstrate the practical application of advanced deep learning techniques in real-world scenarios, providing a foundation for future enhancements such as expanding the dataset, refining model architecture, and deploying the model in practical applications. This project aims to contribute to computer vision and deep learning by showing effective strategies for handling complex and large image classification tasks.

1.3. Scope:

The scope includes meticulous data preparation, sophisticated model architecture design, and the implementation of key callbacks such as EarlyStopping, ReduceLROnPlateau, TensorBoard, and ModelCheckpoint. These components collectively optimize the training process, mitigate overfitting, and ensure the best model performance is retained.

Additionally, the project delves into mixed precision training to improve computational efficiency without sacrificing accuracy. The scope extends to documenting the experiments, outcomes, and challenges encountered, providing valuable insights into the practical application of deep learning in image classification.

Future extensions of this project could involve expanding the dataset to include more diverse food categories, further refining the model architecture, and deploying the model in real-world applications such as restaurant menu analysis, dietary monitoring, and culinary image search engines. By setting a high standard in food image classification, this project paves the way for continued advancements in computer vision and deep learning.

1.4. Objective:

The objective of this project is to develop a high-performance deep learning model for classifying food images into 101 predefined categories, focusing on the following key areas:

1. **Data Preparation and Exploration**: To effectively prepare and explore the Food-101 dataset, ensuring that the data is clean, image cast, well-organized, and ready for model training and evaluation.

2. **Image Transformation and Feature Extraction**: To apply advanced image transformation techniques and utilize EfficientNetB0 for robust feature extraction, enhancing the model's ability to learn from complex food images.

3. **Model Development**: To design and build a deep learning model using EfficientNetB0, incorporating mixed precision training and various callbacks such as EarlyStopping, ReduceLROnPlateau, TensorBoard, and ModelCheckpoint to optimize performance and prevent overfitting.

4. **Model Evaluation and Interpretation**: To rigorously evaluate the model's performance using appropriate metrics and interpret the results to assess accuracy, recall, loss curve, F1 score and other potential areas for improvement.

5. **Comparative Analysis**: To conduct a comparative analysis of the model's performance against baseline and other state-of-the-art models, providing insights into its relative effectiveness and efficiency.

6. **Application and Implications**: To explore practical applications of the model in real-world scenarios, such as restaurant menu analysis, nutritional value checks, and dietary monitoring, and to assess its implications for these applications.

7. **Innovation and Future Work**: To identify opportunities for innovation and outline future work, including expanding the dataset, refining the model to more advanced version of EfficientNet, and exploring new methods for further enhancing classification accuracy.

8. **Documentation and Reporting**: To thoroughly document and report on the project's methodology, experiments, outcomes, and challenges, ensuring transparency and providing valuable insights for future research and development.

1.5. Technology and literature review:

This project utilizes TensorFlow for building and training a deep learning model, with EfficientNetB0 serving as the base architecture. EfficientNetB0 is renowned for its efficient balance of accuracy and computational resources, achieved through a compound scaling method.

The literature highlights key advancements relevant to this project. EfficientNet, introduced by Tan and Le (2019), presents a scalable approach to model design that improves both efficiency and accuracy. Mixed precision training, discussed by Micikevicius et al. (2018), accelerates training by using lower-precision arithmetic. Additionally, practical applications of food image classification are explored in recent reviews, demonstrating the impact of deep learning in real-world scenarios such as dietary monitoring and menu analysis.

2. Problem Statement / Review / Background

2.1. Problem Statement:

In the contemporary digital landscape, the vast increase in food-related imagery across social media platforms, restaurant menus, and recipe blogs has resulted in an extensive collection of food images. Despite this abundance, effectively categorizing these images remains a significant challenge due to the wide variety and intricate nature of food presentations. The ability to accurately classify food images is crucial for improving various applications, including automated dietary tracking, personalized nutrition recommendations, digitalization of restaurant menus, and enhancing food blogging experiences. Addressing this challenge through advanced image classification techniques can transform the way food-related content is managed and utilized, offering more efficient and insightful solutions in these domains.

2.2. Review & Background:

In the contemporary digital age, there has been a dramatic increase in the volume of food-related images available online, spanning social media platforms, restaurant menus, and recipe blogs. This surge has resulted in a vast repository of food images, yet accurately categorizing these diverse and complex presentations remains a significant challenge. The wide variety of food types, styles, and presentation formats complicates the classification process, making it difficult to develop a robust system that can handle such variability.

Accurate classification of food images has substantial implications for various practical applications. Automated dietary tracking systems can benefit from precise food categorization to provide users with accurate nutritional information. Personalized nutrition recommendations can be enhanced by correctly identifying and categorizing food items. Restaurant menu digitization can be streamlined by automatically classifying menu items, and food bloggers can improve content organization and searchability through effective image categorization.

To address these challenges, the project employs advanced deep learning techniques, specifically utilizing the EfficientNetB0 model. EfficientNetB0 is selected for its state-of-the-art performance in image classification tasks, offering a favourable balance between accuracy and computational efficiency. Mixed precision training is used to accelerate model training and reduce memory usage without compromising accuracy. Additionally, the project integrates several callbacks, including EarlyStopping, ReduceLROnPlateau, TensorBoard, and ModelCheckpoint, to optimize training, prevent overfitting, and monitor performance effectively.

Previous research underscores the advancements in deep learning architectures and their application in complex image classification tasks. Notable studies have demonstrated the efficacy of various models and techniques in enhancing classification accuracy and efficiency. This project builds on these advancements, aiming to provide a robust solution for food image classification, with potential benefits across multiple domains, including dietary tracking, nutrition recommendations, and digital content management. By leveraging cutting-edge technology and methodologies, the project seeks to advance the state-of-the-art in food image classification and address the pressing challenges associated with this task.

3. Theory and Datasets

3.1. Theory

In tackling the challenge of food image classification, this project employs a comprehensive approach that combines advanced deep learning techniques with meticulous optimization strategies to achieve high accuracy and efficiency. The core of our approach revolves around the EfficientNetB0 model, renowned for its exceptional performance in image classification tasks. This model leverages a compound scaling method to balance network depth, width, and resolution, resulting in improved accuracy while maintaining computational efficiency.

1. Model Selection and Fine-Tuning:

EfficientNetB0 is chosen as the base model due to its state-of-the-art performance and ability to manage computational resources effectively. To fully harness the potential of this pre-trained model, fine-tuning is applied. Fine-tuning involves adjusting the weights of the pre-trained network through additional training on our specific dataset, allowing the model to learn and adapt to the unique characteristics of food images. By fine-tuning all layers of the EfficientNetB0 model, we refine its ability to classify food images accurately, enhancing its generalization to diverse food types and presentations.

2. Mixed Precision Training:

To accelerate the training process and optimize resource utilization, mixed precision training is implemented. This technique involves using both 16-bit and 32-bit floating-point arithmetic during model training. By leveraging lower-precision arithmetic, mixed precision training reduces memory usage and computational demands, leading to faster training times and more efficient use of hardware resources. Despite the reduction in precision, this technique ensures that model accuracy is preserved, maintaining the reliability of the classification results.

3. Optimization and Callbacks:

- **EarlyStopping**: This callback monitors the model’s performance and halts training when no significant improvement is detected, preventing overfitting and conserving computational resources.
- **ReduceLROnPlateau**: Employed to dynamically adjust the learning rate when the model's performance plateaus, this callback helps facilitate continued learning and convergence.
- **TensorBoard**: Provides real-time visualizations of training metrics, allowing for effective monitoring, debugging, and optimization of the training process.
- **ModelCheckpoint**: This callback saves the model’s weights at specified intervals, ensuring that the best-performing version of the model is preserved throughout training.

By combining these techniques, the approach aims to achieve a high level of classification accuracy while managing computational efficiency. The integration of fine-tuning, mixed precision training, and strategic callbacks reflects a holistic methodology designed to address the complexities of food image classification and deliver robust, reliable results. This approach not only enhances the model’s performance but also sets a strong foundation for future advancements and applications in food image analysis.

3.2. Datasets

Food-101 dataset is a comprehensive and widely utilized benchmark dataset in the field of food image classification. It provides a substantial collection of labeled food images, making it an essential resource for training and evaluating deep learning models in culinary image recognition tasks.

The Food-101 dataset comprises 101 distinct food categories, each representing a different type of food item. The dataset contains a total of 101,000 images, with 1,000 images per category. This large number of images ensures that the dataset is both diverse and comprehensive, covering a wide range of food types and presentation styles.

The images in the Food-101 dataset vary in resolution and aspect ratio, reflecting the real-world diversity of food presentations. Each image is labeled with its corresponding food category, facilitating supervised learning tasks. The dataset includes high-resolution images that are well-suited for training deep learning models, enabling them to capture fine details and subtle differences between food items.

The Food-101 dataset was meticulously curated from various online sources, including food-related websites, recipe blogs, and social media platforms. This diverse collection method ensures that the dataset includes a broad representation of food items from different cuisines and regions. The images were sourced from publicly available content, ensuring a wide variety of food presentations and styles. The diversity in image sources helps in creating a more generalized model that performs well across various food types and presentation styles.

The dataset is organized into a structured directory format, with separate folders for each food category. This organization facilitates easy access and management of the images during the model training and evaluation processes. Preprocessing steps, such as resizing, normalization, and augmentation, are typically applied to prepare the images for input into deep learning models. Common preprocessing techniques include resizing images to a standard dimension, normalizing pixel values.

The Food-101 dataset is widely used in the research and development of food image classification models. Its extensive and diverse collection of food images allows researchers and practitioners to develop and evaluate models with high accuracy and generalization. The dataset serves as a benchmark for comparing the performance of various image classification algorithms and contributes to advancing the state-of-the-art in food image recognition.

While the Food-101 dataset is a valuable resource, it also presents certain challenges. The variability in image quality, lighting conditions, and food presentation can affect model performance. Additionally, the dataset's coverage of food items is limited to 101 categories, which may not encompass all possible food types or regional variations. These factors must be considered when designing models and evaluating their performance.

Overall, the Food-101 dataset provides a robust foundation for developing and testing food image classification models. Its extensive collection of labeled images, diverse food categories, and practical relevance make it an essential tool for advancing research and applications in culinary image recognition.

4. Implementation Details

1. Data Preprocessing and Exploration

1. **Image Size and Channels**:
   - Each image in the dataset is resized to 224 x 224 pixels to match the input size required by the model. This resizing ensures that all images have a uniform dimension, which is crucial for consistent model performance.
   - After resizing, the image data is cast to the `float32` data type. This conversion standardizes the data type across all images, ensuring compatibility with the model's input requirements.
2. **Data Pipeline for Training and Testing:**
   - The training dataset is processed using the `map` function, which applies the `preprocess_img` function to each image. This function resizes and casts the images, and the dataset is then shuffled with a buffer size of 1000 to randomize the order of the images. Shuffling helps in reducing bias and improving model generalization.
   - The training data is batched with a batch size of 32, and the `prefetch` function is used to optimize data loading by overlapping data preparation with model training. This approach speeds up the training process by reducing data loading times.
   - Similarly, the testing dataset is processed with the same `preprocess_img` function, but without shuffling. It is batched with a batch size of 32 and prefetched to ensure efficient data handling during evaluation.

2. Model Architecture

1. **Feature Extraction**:
   - **Base Model**: EfficientNetB0 is utilized for feature extraction. It is known for its efficient performance and accuracy in image classification tasks.
   - **Pre-trained Weights**: The model uses pre-trained weights from ImageNet to leverage learned features, which enhances performance on food category classification.
2. **Global Average Pooling**:
   - **Layer**: A Global Average Pooling 2D layer is used to reduce the spatial dimensions of the feature maps. This operation calculates the average output of each feature map, resulting in a single vector per feature map.
3. **Dense Layers**:
   - **Output Layer**: A Dense layer with 101 units corresponds to the number of food categories in the dataset. It acts as the classifier for the final predictions.
4. **Activation Function**:
   - **Softmax**: The output of the Dense layer is passed through a softmax activation function, which provides probabilities for each food category, facilitating multi-class classification.

3. Mixed Precision Training

1. **Objective**:
    - Mixed precision training is employed to accelerate model training and reduce memory usage by using both 16-bit and 32-bit floating-point types.
2. **Implementation**:
    - **TensorFlow Support**: Utilize TensorFlow’s mixed precision API, which helps in automatic casting of variables and computations to the appropriate precision.
    - **Setup**: Enable mixed precision by setting the policy to 'mixed_float16', which instructs TensorFlow to use float16 precision where possible, while maintaining float32 precision for certain operations to ensure numerical stability.
3. **Benefits**:
    - Improved training speed and reduced memory consumption are significant advantages of using mixed precision. This allows for larger batch sizes and more efficient computation.

4. Training Optimization

1. **Callbacks**:
    - **ReduceLROnPlateau**: Adjusts the learning rate when a plateau in model performance is detected, helping in fine-tuning the learning process.
    - **TensorBoard**: Provides visualization tools to monitor model training progress, including metrics like loss and accuracy.
    - **ModelCheckpoint**: Saves the model at regular intervals, ensuring that the best model weights are preserved.
    - **EarlyStopping**: Monitors validation performance and stops training early if no improvement is observed, preventing overfitting.
2. **Loss Function and Metrics**:
    - **Loss Function**: Loss function is typically used for multi-class classification tasks.
    - **Metrics**: Accuracy is monitored to gauge model performance.

5. Explanation of the Source code

Explanation of the code for the project as below:

1. Data Collection and Initial Exploration

This section begins by mounting Google Drive to access project files, ensuring seamless data retrieval. Essential libraries like Pandas, NumPy, Matplotlib are imported for data manipulation, analysis, and visualization. The code loads the raw dataset from a CSV file containing images and associated features. We are using the Food-101 dataset, which contains images of food items categorized into 101 different classes. This dataset is particularly useful for training models to recognize various food items.

2. Visualizing the sample image

In this part, we begin by taking a single sample from the training dataset. This allows us to inspect and visualize the data we're working with. Using a loop, we extract the image and its corresponding label from the sample. We print out the shape and data type of the image tensor, which helps us understand the input dimensions and format required by our model. We also convert the label tensor to a human-readable class name using the class_names list.

3. Data Preprocessing

We define a preprocessing function to ensure that all images are resized to a uniform size of 224x224 pixels and are cast to a float32 data type. This uniformity is essential for consistent input to the neural network. We then apply this preprocessing function to both our training and testing datasets. For the training data, we also shuffle the data to break any inherent order, which helps in better generalization by the model. The data is then batched into groups of 32 samples. Batching helps in efficient processing and speeds up the training by allowing the model to process multiple samples simultaneously. Finally, we use prefetching to ensure that data is loaded in the background while the model is training on the current batch. This reduces idle time and improves the overall training efficiency.

4. Define Model and Mixed Precision Training

We set the mixed precision policy to use both 16-bit and 32-bit floating-point types. This helps in speeding up the training process and reducing memory usage, which is particularly beneficial when training large models. We use the EfficientNetB0 model as our base model. EfficientNetB0 is known for its efficiency in terms of both parameters and FLOPS. We exclude the top layers of this pre-trained model to add our custom layers for the specific task of food image classification. The base model's weights are frozen initially to leverage the pre-trained features and prevent overfitting during the initial training phase.

Our model starts with an input layer that matches the shape of our images (224x224x3). The images are passed through the EfficientNetB0 base model for feature extraction. We then apply global average pooling to reduce the spatial dimensions of the feature maps. This is followed by a dense layer with 101 units, corresponding to the number of food categories. Finally, we apply a softmax activation function to convert the logits into probabilities for each class. We compile the model using sparse categorical cross-entropy as the loss function, which is suitable for our multi-class classification task where labels are integers.

We use the Adam optimizer for training, and we track accuracy as our primary metric. The model summary provides a detailed overview of our architecture, showing the layers, output shapes, and the number of parameters. This helps in understanding the complexity and size of our model.

5. Define Callbacks

We use callbacks to perform specific actions during training. In our case, we use the ModelCheckpoint callback to save the model's weights whenever the validation accuracy improves. This ensures that we always have the best version of the model saved. The checkpoint path is defined as model_checkpoints/cp.weights.h5, and we monitor the validation accuracy to decide when to save the model weights. By setting save_best_only=True and save_weights_only=True, we ensure that only the best model weights are saved, without the entire model architecture, which saves storage space.

We use two callbacks during training. The first is a custom TensorBoard callback to log training metrics for visualization. The second is the ModelCheckpoint callback to save the model weights whenever the validation accuracy improves. TensorBoard callback logs various metrics such as loss and accuracy during training. These logs can be visualized using TensorBoard, which provides a comprehensive view of the training process, helping us understand how well the model is learning. The ModelCheckpoint callback ensures that we always save the best-performing model based on validation accuracy. This is crucial for selecting the best model version for further evaluation and deployment.

6. Model Evaluation

After training our model, the next step is to evaluate its performance on the test dataset. This dataset consists of samples that the model has never seen before, providing an unbiased estimate of how well the model generalizes to new data. We use the model.evaluate() method to assess the model's performance. This method computes the loss and accuracy on the test dataset. The loss gives us an idea of how well the model's predictions align with the actual labels, while the accuracy indicates the proportion of correct predictions. In our code, we pass the test dataset to the model.evaluate() method, which returns the loss and accuracy. We then print these results to understand the model's performance. Evaluating the model on unseen data is crucial to ensure that it performs well in real-world scenarios. High accuracy on the test set indicates that the model has effectively learned to generalize from the training data. By comparing the evaluation results with the training and validation metrics, we can also identify if the model is overfitting (performing well on training data but poorly on test data) or underfitting (performing poorly on both training and test data). For example, if the printed evaluation results are something like [1.000953197479248, 0.7278019785881042], this means that the loss on the test dataset is approximately 1.00095, and the accuracy is approximately 72.78%. This indicates that the model is correctly classifying about 72.78% of the test samples.

7. Loss & Accuracy curve visualization

Visualizing these metrics is a crucial step in the model development process, providing valuable feedback and guiding us towards creating a robust and well-generalized model. After training our model, it's crucial to visualize the performance metrics to understand how well our model is learning over time. By plotting the training and validation loss and accuracy, we can gain insights into the model's learning process and identify potential issues such as overfitting or underfitting. We plot the training and validation loss over epochs. This helps us see how the model's loss decreases over time, indicating how well the model is minimizing the error during training. Ideally, both the training and validation loss should decrease and stabilize, indicating good learning. We plot the training and validation accuracy over epochs. This shows how the model's accuracy improves as it learns from the data. We aim for both the training and validation accuracy to increase and converge, indicating the model is generalizing too well. Thus, we identify that the model is overfitting.

8. Confusion Matrix & Classification Report

The confusion matrix is a powerful tool for evaluating the performance of our classification model, providing detailed insights into its strengths and weaknesses. This visualization helps us identify specific areas for improvement, guiding further model tuning and optimization. By examining classification report, we can identify which classes our model performs well on and which ones need improvement. High precision and recall values indicate good performance, while lower values indicate areas where the model may be struggling.

9. Fine-Tuning the model

After initial training, we proceed to fine-tune the model. Fine-tuning allows the model to adapt more specifically to our dataset by unfreezing the layers and continuing training. We start by loading the weights from the best model checkpoint saved during initial training. Then, we set all the layers of the model to be trainable, allowing the model to update the weights of all layers during fine-tuning. We use several callbacks to optimize the fine-tuning process. EarlyStopping halts training if the validation loss doesn't improve for 3 epochs, preventing overfitting. ReduceLROnPlateau reduces the learning rate if the validation loss plateaus, helping the model converge. ModelCheckpoint saves the best model based on validation loss, ensuring we keep the best-performing version.

We then fine-tune the model for up to 100 epochs, using the training data and evaluating on the validation data. The training process is monitored using TensorBoard for visualization, and the best model is saved based on validation loss. EarlyStopping and ReduceLROnPlateau help optimize the training process by preventing overfitting and adjusting the learning rate. Fine-tuning leverages the pre-trained knowledge of the model while allowing it to adapt more specifically to our dataset, leading to improved performance. The use of callbacks ensures that the training process is efficient and effective.

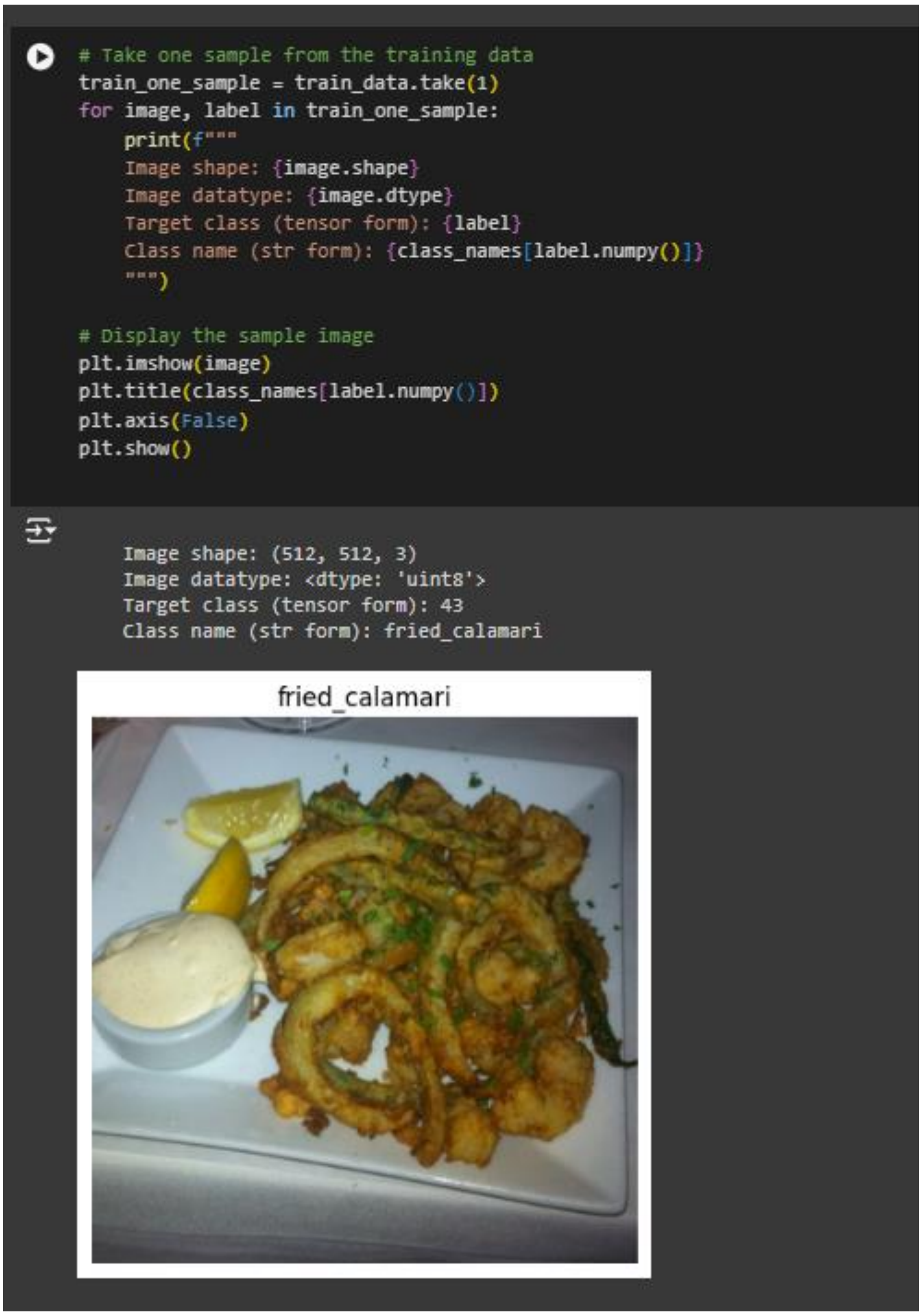
10. Predicting the food image after fine-tuning

The image is first opened using the Python Imaging Library (PIL). This library is used for various image processing tasks and provides a straightforward way to handle image files. Once the image is opened, it is converted into a NumPy array. This format is necessary for further processing with TensorFlow. The NumPy array is then converted into a TensorFlow tensor. This conversion allows the image data to be used in TensorFlow's computational framework. A batch dimension is added to the tensor. This adjustment is required because the model expects inputs in batches, even if it's a single image. The image is resized to 224 x 224 pixels to match the input size required by the model. Additionally, the image tensor is cast to the float32 data type to ensure compatibility with the model's expectations. The pre-trained and fine-tuned model, which has been trained on the Food-101 dataset, is loaded. This model is designed to classify images into one of the 101 food categories. The predicted class index is then used to determine the corresponding food category label from the predefined set of categories.

This project successfully uses the Food-101 dataset to illustrate the application of deep learning for food image classification. The EfficientNetB0 model, when combined with effective preprocessing, data augmentation, and training procedures, produces promising results. The challenges encountered provide insights for future developments, paving the door for greater advancements in food image classification. The proposed model can be used to support a variety of food industry applications, such as nutritional tracking, automated recipe recommendation, and culinary instruction. Future work will concentrate on fine-tuning the model and investigating new ways to attain even greater accuracy and robustness.

6. Results and Discussion

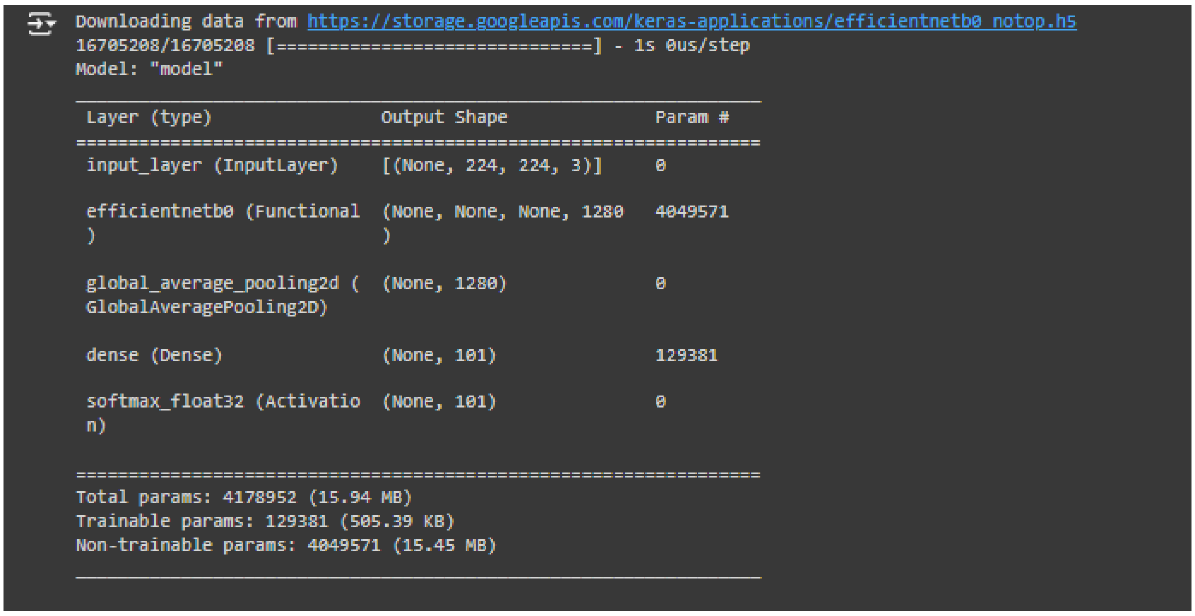
➔Inspect a sample of image from the dataset.



```python
# Take one sample from the training data
train_one_sample = train_data.take(1)
for image, label in train_one_sample:
    print(f"""
    Image shape: {image.shape}
    Image datatype: {image.dtype}
    Target class (tensor form): {label}
    Class name (str form): {class_names[label.numpy()]}
    """)

# Display the sample image
plt.imshow(image)
plt.title(class_names[label.numpy()])
plt.axis(False)
plt.show()
```

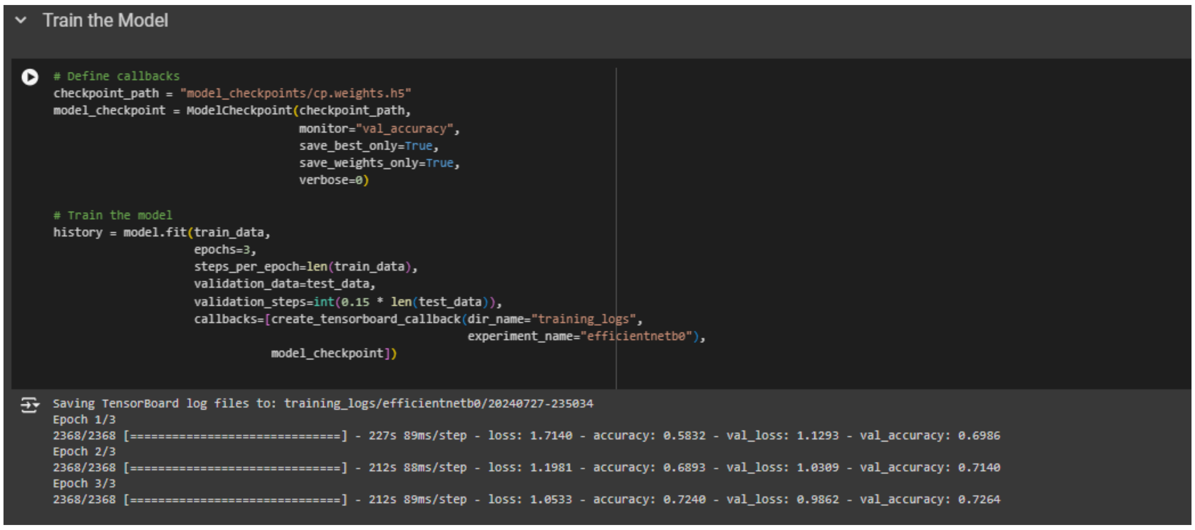
```
    Image shape: (512, 512, 3)
    Image datatype: <dtype: 'uint8'>
    Target class (tensor form): 43
    Class name (str form): fried_calamari
```

➔Build the model and image of architecture of model.



```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 1s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 efficientnetb0 (Functional  (None, None, None, 1280   4049571
 )                           )

 global_average_pooling2d (  (None, 1280)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 101)               129381

 softmax_float32 (Activatio  (None, 101)               0
 n)

=================================================================
Total params: 4178952 (15.94 MB)
Trainable params: 129381 (505.39 KB)
Non-trainable params: 4049571 (15.45 MB)
_________________________________________________________________
```

➔ Train the model with 3 epochs.

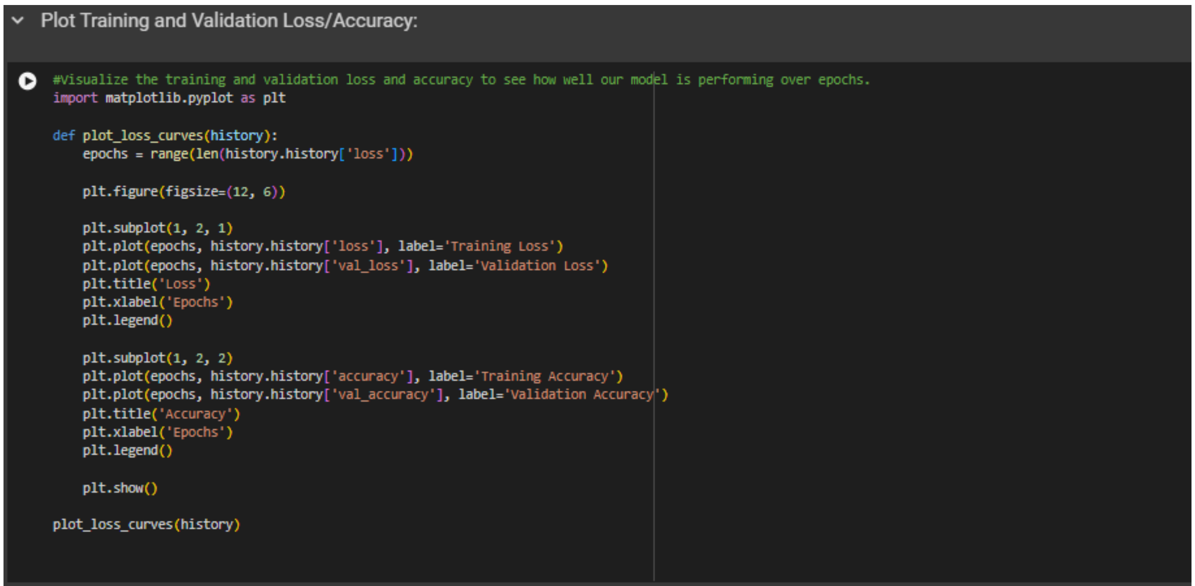


Train the Model

```
# Define callbacks
checkpoint_path = "model_checkpoints/cp.weights.h5"
model_checkpoint = ModelCheckpoint(checkpoint_path,
                                   monitor="val_accuracy",
                                   save_best_only=True,
                                   save_weights_only=True,
                                   verbose=0)


# Train the model
history = model.fit(train_data,
                    epochs=3,
                    steps_per_epoch=len(train_data),
                    validation_data=test_data,
                    validation_steps=int(0.15 * len(test_data)),
                    callbacks=[create_tensorboard_callback(dir_name="training_logs",
                                                           experiment_name="efficientnetb0"),
                               model_checkpoint])
```

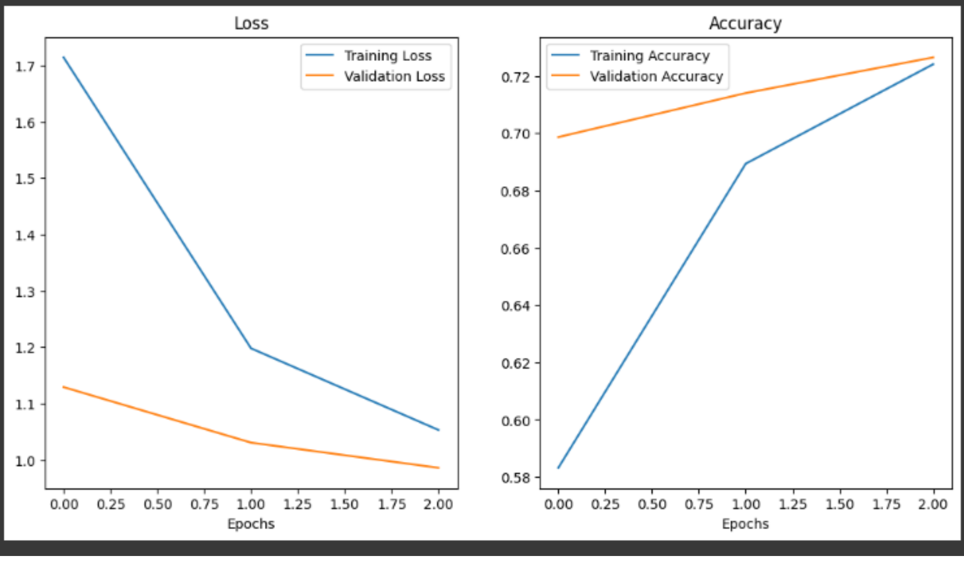
```
Saving TensorBoard log files to: training_logs/efficientnetb0/20240727-235034
Epoch 1/3
2368/2368 [==============================] - 227s 89ms/step - loss: 1.7140 - accuracy: 0.5832 - val_loss: 1.1293 - val_accuracy: 0.6986
Epoch 2/3
2368/2368 [==============================] - 212s 88ms/step - loss: 1.1981 - accuracy: 0.6893 - val_loss: 1.0309 - val_accuracy: 0.7140
Epoch 3/3
2368/2368 [==============================] - 212s 89ms/step - loss: 1.0533 - accuracy: 0.7240 - val_loss: 0.9862 - val_accuracy: 0.7264
```

➔ Plot the Loss & Accuracy curve

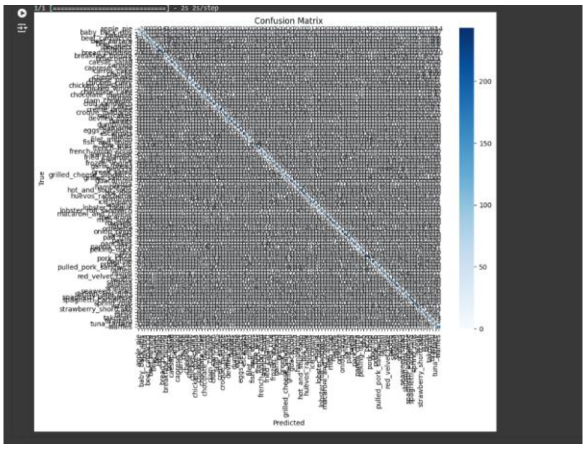


Plot Training and Validation Loss/Accuracy:

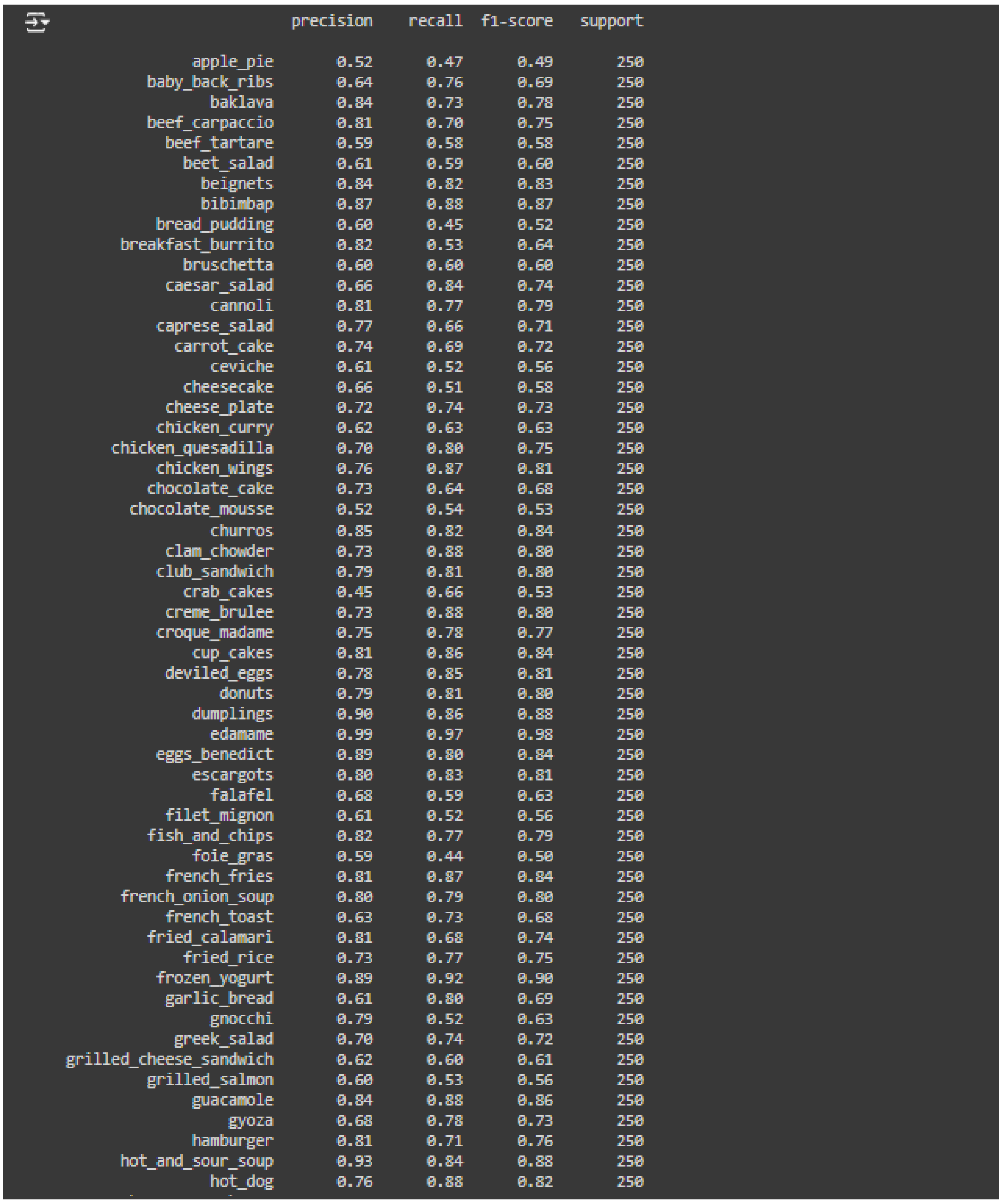
```
#Visualize the training and validation loss and accuracy to see how well our model is performing over epochs.
import matplotlib.pyplot as plt

def plot_loss_curves(history):
    epochs = range(len(history.history['loss']))

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, history.history['loss'], label='Training Loss')
    plt.plot(epochs, history.history['val_loss'], label='Validation Loss')
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history.history['accuracy'], label='Training Accuracy')
    plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.legend()

    plt.show()

plot_loss_curves(history)
```



➔ Calculate confusion matrix.

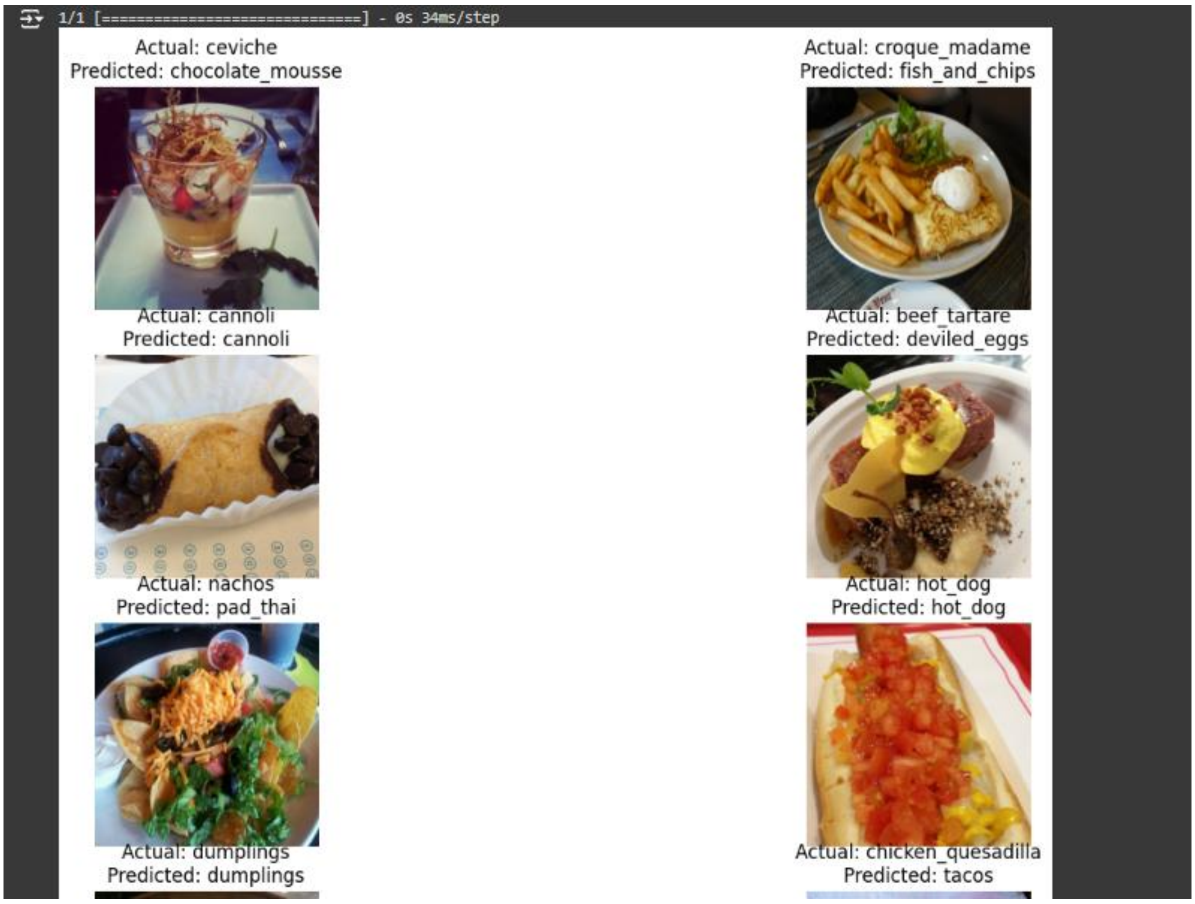


➔ Shown the classification report.



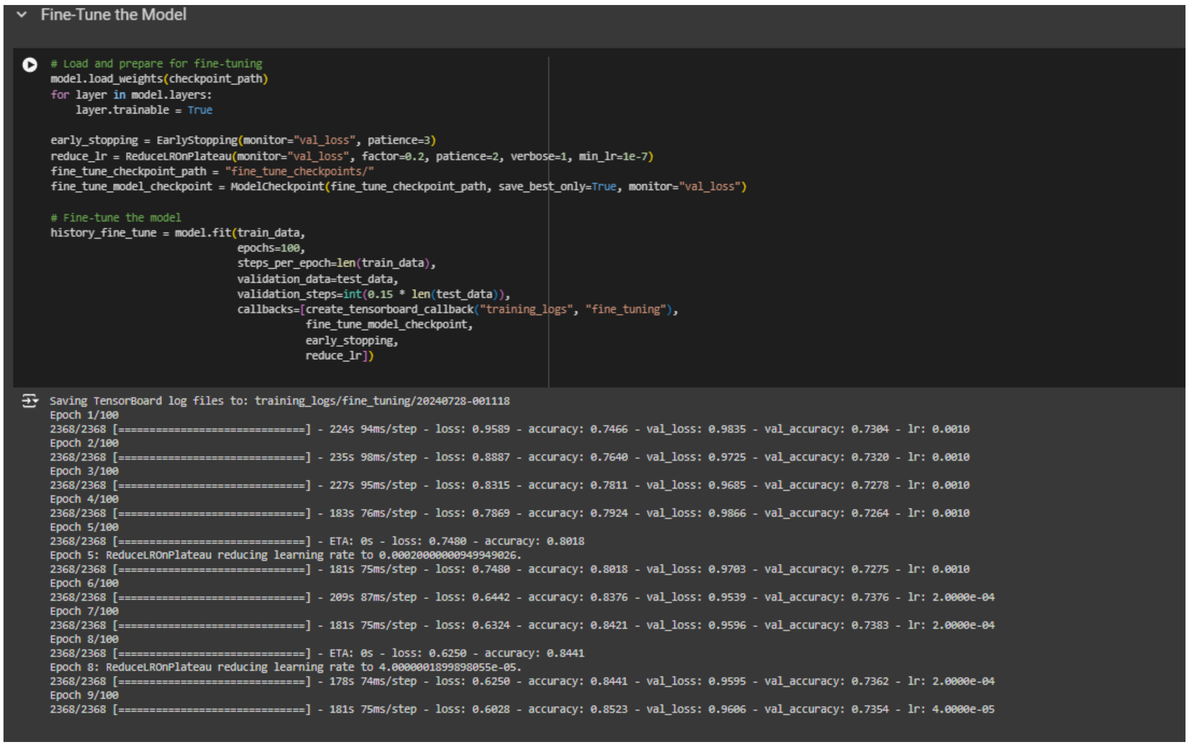
```
                         precision    recall  f1-score   support

              apple_pie       0.52      0.47      0.49       250
         baby_back_ribs       0.64      0.76      0.69       250
                baklava       0.84      0.73      0.78       250
         beef_carpaccio       0.81      0.70      0.75       250
            beef_tartare       0.59      0.58      0.58       250
             beet_salad       0.61      0.59      0.60       250
               beignets       0.84      0.82      0.83       250
               bibimbap       0.87      0.88      0.87       250
          bread_pudding       0.60      0.45      0.52       250
      breakfast_burrito       0.82      0.53      0.64       250
             bruschetta       0.60      0.60      0.60       250
            caesar_salad       0.66      0.84      0.74       250
                cannoli       0.81      0.77      0.79       250
          caprese_salad       0.77      0.66      0.71       250
            carrot_cake       0.74      0.69      0.72       250
                ceviche       0.61      0.52      0.56       250
             cheesecake       0.66      0.51      0.58       250
           cheese_plate       0.72      0.74      0.73       250
          chicken_curry       0.62      0.63      0.63       250
     chicken_quesadilla       0.70      0.80      0.75       250
          chicken_wings       0.76      0.87      0.81       250
         chocolate_cake       0.73      0.64      0.68       250
       chocolate_mousse       0.52      0.54      0.53       250
                churros       0.85      0.82      0.84       250
           clam_chowder       0.73      0.88      0.80       250
          club_sandwich       0.79      0.81      0.80       250
             crab_cakes       0.45      0.66      0.53       250
           creme_brulee       0.73      0.88      0.80       250
          croque_madame       0.75      0.78      0.77       250
              cup_cakes       0.81      0.86      0.84       250
           deviled_eggs       0.78      0.85      0.81       250
                 donuts       0.79      0.81      0.80       250
              dumplings       0.90      0.86      0.88       250
                edamame       0.99      0.97      0.98       250
          eggs_benedict       0.89      0.80      0.84       250
              escargots       0.80      0.83      0.81       250
                falafel       0.68      0.59      0.63       250
           filet_mignon       0.61      0.52      0.56       250
         fish_and_chips       0.82      0.77      0.79       250
              foie_gras       0.59      0.44      0.50       250
           french_fries       0.81      0.87      0.84       250
      french_onion_soup       0.80      0.79      0.80       250
           french_toast       0.63      0.73      0.68       250
         fried_calamari       0.81      0.68      0.74       250
             fried_rice       0.73      0.77      0.75       250
          frozen_yogurt       0.89      0.92      0.90       250
           garlic_bread       0.61      0.80      0.69       250
                gnocchi       0.79      0.52      0.63       250
            greek_salad       0.70      0.74      0.72       250
grilled_cheese_sandwich       0.62      0.60      0.61       250
         grilled_salmon       0.60      0.53      0.56       250
              guacamole       0.84      0.88      0.86       250
                  gyoza       0.68      0.78      0.73       250
              hamburger       0.81      0.71      0.76       250
      hot_and_sour_soup       0.93      0.84      0.88       250
                hot_dog       0.76      0.88      0.82       250
```

➔ Shows the best wrong predictions after training the model.



```
1/1 [==============================] - 0s 34ms/step
```

➔ Fine-Tune the model

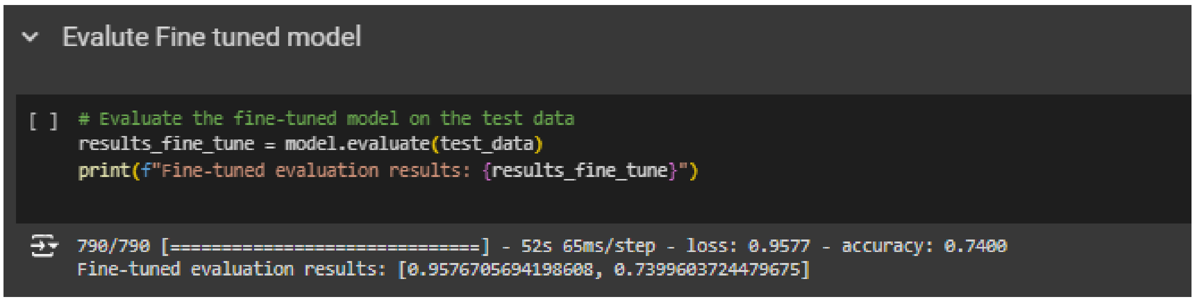


Fine-Tune the Model

```python
# Load and prepare for fine-tuning
model.load_weights(checkpoint_path)
for layer in model.layers:
    layer.trainable = True

early_stopping = EarlyStopping(monitor="val_loss", patience=3)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=2, verbose=1, min_lr=1e-7)
fine_tune_checkpoint_path = "fine_tune_checkpoints/"
fine_tune_model_checkpoint = ModelCheckpoint(fine_tune_checkpoint_path, save_best_only=True, monitor="val_loss")

# Fine-tune the model
history_fine_tune = model.fit(train_data,
                              epochs=100,
                              steps_per_epoch=len(train_data),
                              validation_data=test_data,
                              validation_steps=int(0.15 * len(test_data)),
                              callbacks=[create_tensorboard_callback("training_logs", "fine_tuning"),
                                         fine_tune_model_checkpoint,
                                         early_stopping,
                                         reduce_lr])
```

```
Saving TensorBoard log files to: training_logs/fine_tuning/20240728-001118
Epoch 1/100
2368/2368 [==============================] - 224s 94ms/step - loss: 0.9589 - accuracy: 0.7466 - val_loss: 0.9835 - val_accuracy: 0.7304 - lr: 0.0010
Epoch 2/100
2368/2368 [==============================] - 235s 98ms/step - loss: 0.8887 - accuracy: 0.7640 - val_loss: 0.9725 - val_accuracy: 0.7320 - lr: 0.0010
Epoch 3/100
2368/2368 [==============================] - 227s 95ms/step - loss: 0.8315 - accuracy: 0.7811 - val_loss: 0.9685 - val_accuracy: 0.7278 - lr: 0.0010
Epoch 4/100
2368/2368 [==============================] - 183s 76ms/step - loss: 0.7869 - accuracy: 0.7924 - val_loss: 0.9866 - val_accuracy: 0.7264 - lr: 0.0010
Epoch 5/100
2368/2368 [==============================] - ETA: 0s - loss: 0.7480 - accuracy: 0.8018
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
2368/2368 [==============================] - 181s 75ms/step - loss: 0.7480 - accuracy: 0.8018 - val_loss: 0.9703 - val_accuracy: 0.7275 - lr: 0.0010
Epoch 6/100
2368/2368 [==============================] - 209s 87ms/step - loss: 0.6442 - accuracy: 0.8376 - val_loss: 0.9539 - val_accuracy: 0.7376 - lr: 2.0000e-04
Epoch 7/100
2368/2368 [==============================] - 181s 75ms/step - loss: 0.6324 - accuracy: 0.8421 - val_loss: 0.9596 - val_accuracy: 0.7383 - lr: 2.0000e-04
Epoch 8/100
2368/2368 [==============================] - ETA: 0s - loss: 0.6250 - accuracy: 0.8441
Epoch 8: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
2368/2368 [==============================] - 178s 74ms/step - loss: 0.6250 - accuracy: 0.8441 - val_loss: 0.9595 - val_accuracy: 0.7362 - lr: 2.0000e-04
Epoch 9/100
2368/2368 [==============================] - 181s 75ms/step - loss: 0.6028 - accuracy: 0.8523 - val_loss: 0.9606 - val_accuracy: 0.7354 - lr: 4.0000e-05
```

➔ Evaluation of Fine-Tuned model.

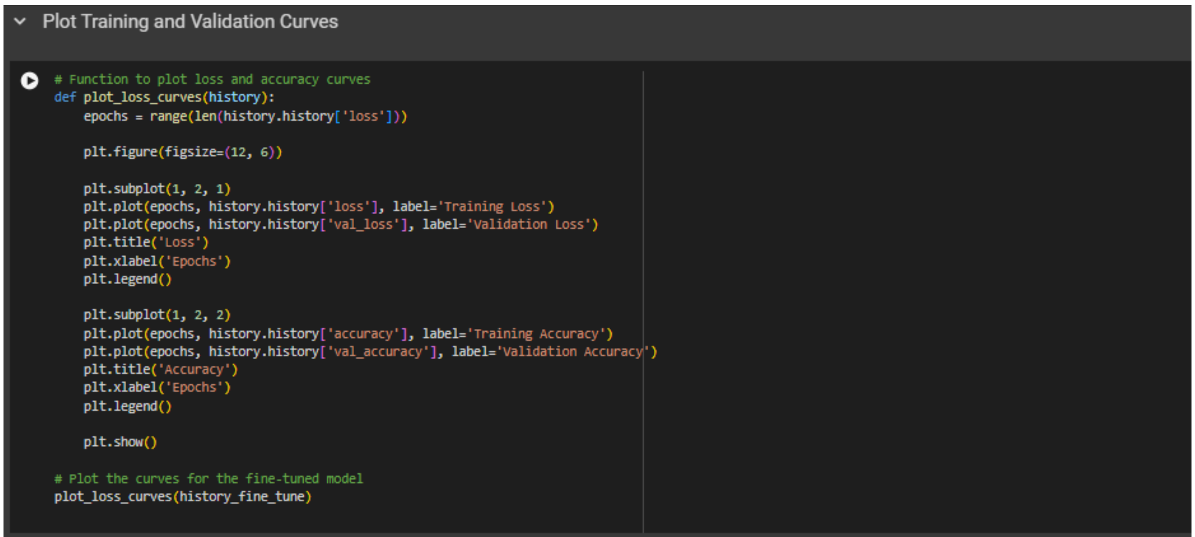


Evalute Fine tuned model

```
[ ] # Evaluate the fine-tuned model on the test data
    results_fine_tune = model.evaluate(test_data)
    print(f"Fine-tuned evaluation results: {results_fine_tune}")
```

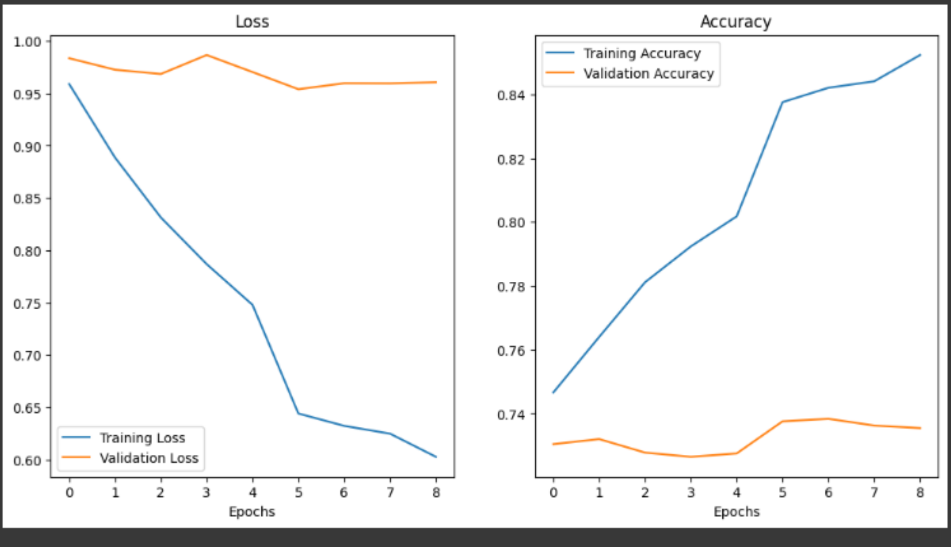
```
790/790 [==============================] - 52s 65ms/step - loss: 0.9577 - accuracy: 0.7400
Fine-tuned evaluation results: [0.9576705694198608, 0.7399603724479675]
```

➔ Plot the training and validation loss and accuracy after fine-tuning.

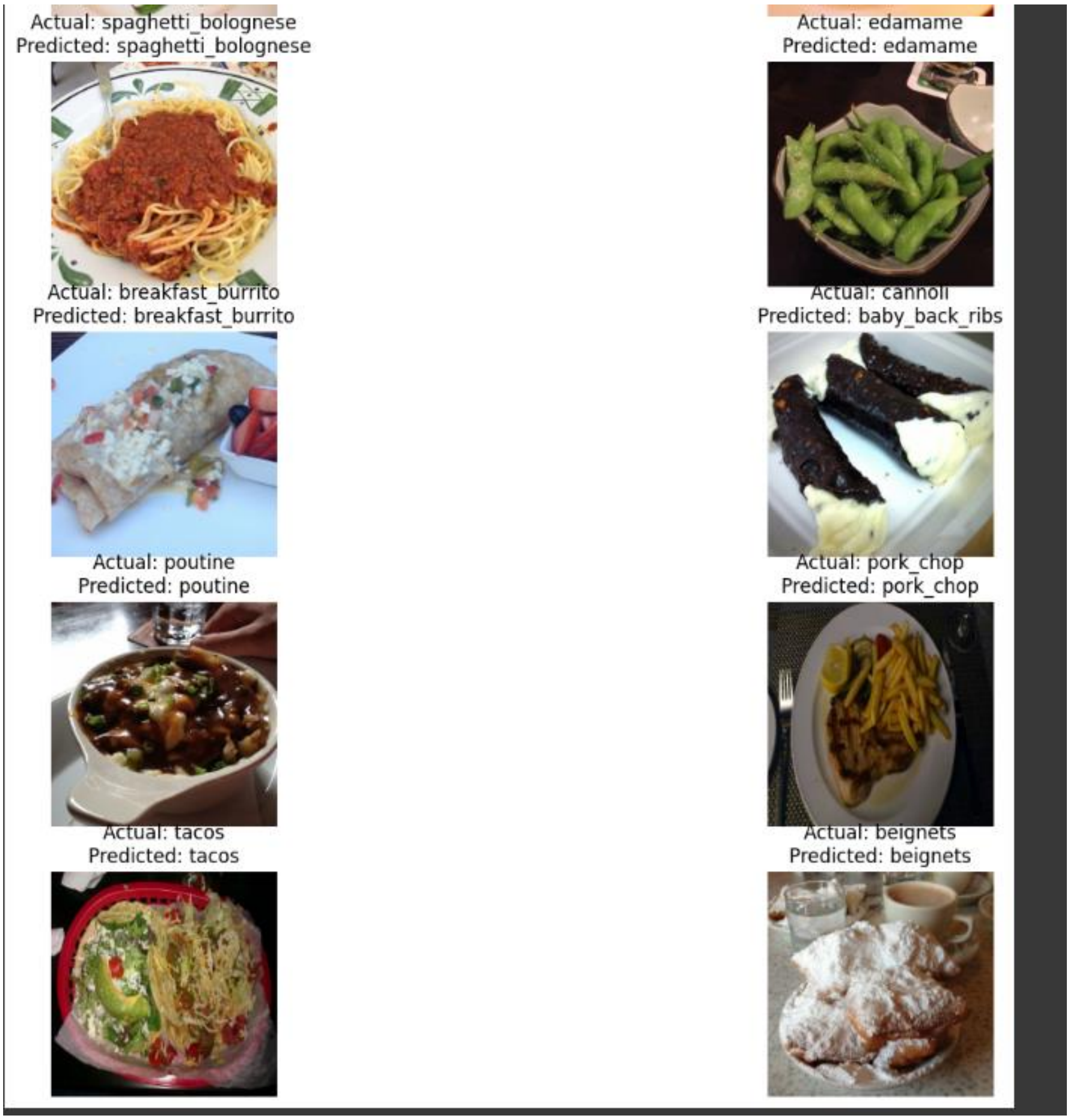


Plot Training and Validation Curves

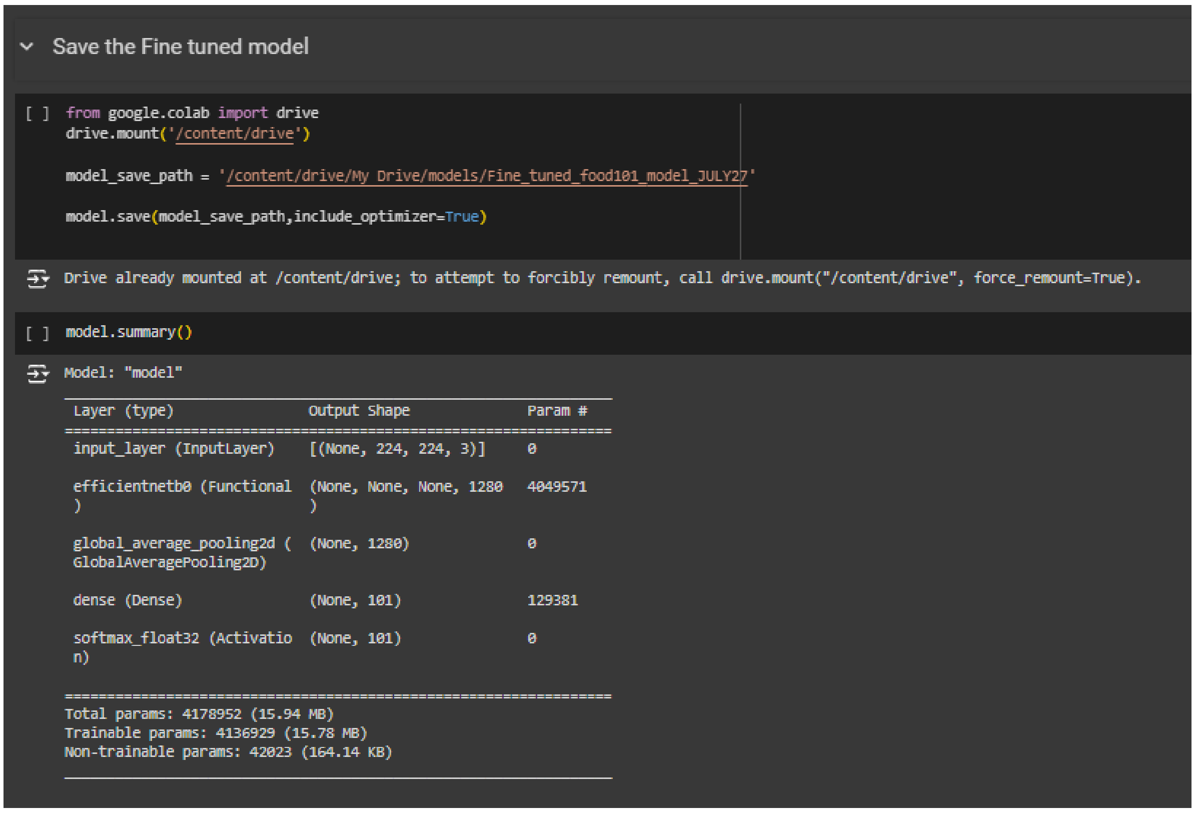
```
# Function to plot loss and accuracy curves
def plot_loss_curves(history):
    epochs = range(len(history.history['loss']))

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, history.history['loss'], label='Training Loss')
    plt.plot(epochs, history.history['val_loss'], label='Validation Loss')
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history.history['accuracy'], label='Training Accuracy')
    plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.legend()

    plt.show()

# Plot the curves for the fine-tuned model
plot_loss_curves(history_fine_tune)
```



➔ Generated the confusion matrix and classification report for the model after fine-tuning the model and predicted some of the examples as well.



➔ Saved the final fine-tuned model and predict the output after uploading image to predict.



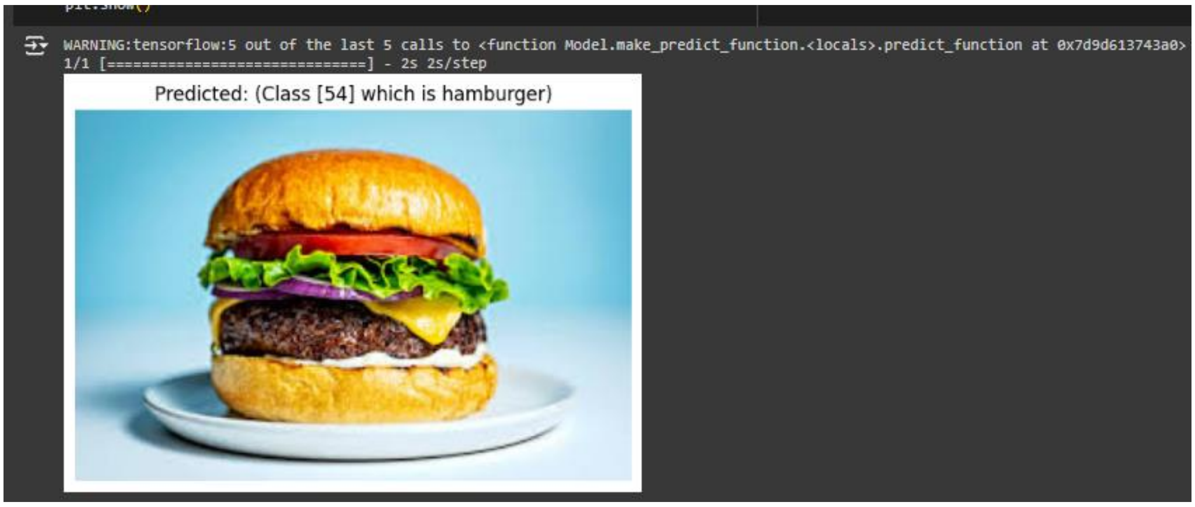
Save the Fine tuned model

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    model_save_path = '/content/drive/My Drive/models/Fine_tuned_food101_model_JULY27'

    model.save(model_save_path,include_optimizer=True)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 efficientnetb0 (Functional  (None, None, None, 1280   4049571
 )                           )

 global_average_pooling2d (  (None, 1280)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 101)               129381

 softmax_float32 (Activatio  (None, 101)               0
 n)

=================================================================
Total params: 4178952 (15.94 MB)
Trainable params: 4136929 (15.78 MB)
Non-trainable params: 42023 (164.14 KB)
_________________________________________________________________
```



```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7d9d613743a0>
1/1 [==============================] - 2s 2s/step
```

After fine-tuning the deep learning model, we achieved a training accuracy of 84%, which represents a significant improvement over the pre-fine-tuning accuracy of approximately 72.78%. However, the validation accuracy remains at 74%, indicating that while the model has improved on the training data, there is still a noticeable gap in its performance on unseen data. This disparity suggests that further enhancements are needed to boost the model's generalization capability. Despite these improvements, there is considerable room for further development. Addressing the challenges of overfitting and enhancing the model's ability to generalize across diverse food categories will be crucial for achieving higher validation accuracy and overall model effectiveness.

7. Recommendations for Future work

7.1. Limitations:

While this project aims to develop an accurate and efficient image classification, some inherent limitations need to be acknowledged:

1. Overfitting: Overfitting was a significant challenge throughout the project. Despite implementing data augmentation and regularization techniques to mitigate this issue, the model's performance still showed signs of overfitting. The complexity of the deep learning model and the variability in the training data contributed to this challenge.
2. Variety in Food Presentation and Lighting: The diversity in food presentation and lighting conditions presented substantial difficulties. The model struggled with variations in how food items were presented and captured in different lighting environments. This variability required robust preprocessing and augmentation strategies to enhance the model's ability to generalize across various scenarios.
3. Training Accuracy**:** The training accuracy achieved by the model was 84%, which, although acceptable, is not exceptionally high for this type of classification task. This level of accuracy indicates that there is room for improvement, and the model may benefit from further refinement, additional data, or enhanced feature extraction techniques.
4. Data Limitations: The Food-101 dataset, while extensive, may still contain biases or gaps in representation. Certain food categories or presentation styles might be underrepresented, impacting the model’s overall performance and its ability to accurately classify less common categories.
5. Computational Resources: Training deep learning models, especially with complex architectures like EfficientNetB0, requires significant computational resources. The availability of such resources can be a limiting factor, affecting the ability to experiment with larger models or more extensive hyperparameter tuning.

7.2. Future enhancements:

To overcome the aforementioned limitations and further improve performance, several future enhancements can be considered:

1. Model Architecture Improvements: Explore more advanced or recent architectures beyond EfficientNetB0, such as EfficientNetV2, Vision Transformers (ViTs), or convolutional neural networks (CNNs) with attention mechanisms. Combine multiple models to leverage their diverse strengths and improve overall performance. An ensemble of models can help in achieving higher accuracy and robustness.
2. Data Augmentation: Implement additional data augmentation techniques such as geometric transformations, color jittering, and more complex synthetic data generation to improve model generalization. Increase the size and diversity of the training dataset by incorporating more images or using data from other sources. This can help in covering a broader range of food presentations and lighting conditions.
3. Model Regularization: Explore advanced regularization methods like dropout, L2 regularization, and batch normalization to further reduce overfitting and improve model generalization.
4. Incorporation of Additional Features: Integrate additional data modalities, such as textual descriptions or nutritional information, to enhance the model's understanding and classification capabilities. Use contextual information, such as the background or setting of the image, to improve classification accuracy.

8. Conclusion

8.1. Conclusion

This project exemplifies the transformative impact of fine-tuning deep learning models for specialized tasks, highlighting the practical potential of artificial intelligence in real-world applications. Throughout this project, we developed and meticulously fine-tuned a deep learning model tailored to classify food images from the comprehensive Food-101 dataset. By leveraging the EfficientNetB0 architecture, known for its efficient performance and accuracy, we created a robust and reliable model capable of distinguishing between 101 distinct food categories.

The fine-tuning process allowed us to adapt a pre-trained model to our specific domain, significantly enhancing its performance for food image classification. This approach not only improved the model’s accuracy but also demonstrated the effectiveness of transfer learning in tackling specialized tasks. Our model’s capability to accurately categorize a wide range of food items underscores the power of deep learning techniques in understanding and interpreting complex visual data.

The potential applications of this model are extensive and impactful, particularly in the food industry. For nutritional tracking, the model's accurate classification of food items can significantly aid in monitoring nutritional intake, offering valuable insights for health-conscious individuals. In dietary planning, the model can facilitate the creation of personalized dietary plans by identifying food categories and suggesting appropriate alternatives or adjustments based on specific dietary needs. Additionally, the model's classification capabilities can be leveraged for automated recipe recommendations, where it helps systems suggest recipes based on the identified ingredients or food items. In the realm of culinary instruction, the model can enhance learning and teaching by providing detailed information about various food items, making it a useful tool for both educational and professional purposes.

In summary, the successful development and fine-tuning of this deep learning model not only showcase the capabilities of modern AI techniques but also pave the way for innovative applications in the food industry. The project highlights how advanced machine learning models can be harnessed to address real-world challenges and provide meaningful solutions in various domains.

9. References

1. L. BOSSARD, M. GUILLAUMIN, AND L. VAN GOOL, “FOOD-101 – MINING DISCRIMINATIVE COMPONENTS WITH RANDOM FORESTS,” IN PROC. OF THE EUROPEAN CONFERENCE ON COMPUTER VISION (ECCV), 2014.
2. K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
3. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in Proc. of the Advances in Neural Information Processing Systems (NIPS), 2012.
4. M. Khosravi and B. Farzad, “Image Classification Using Deep Learning: A Comprehensive Review,” Multimedia Tools and Applications, vol. 80, no. 6, pp. 8423-8458, March 2021.
   Smartphone-based food recognition system using multiple deep CNN models | Multimedia Tools and Applications (springer.com)