A blue text on a black background

AI-generated content may be incorrect.

**School of Computing**

**COMP8260 AI Systems Implementation**

**AI System Project**

**Food Image Classification for Nutritional Estimation**

**Instructor Team:**

Dr. Matteo Migliavacca

George Ramzi

**Submission Date:** 28th March 2025

**Group Name:** P01

**Team Members:**

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Name** | **Email** |
| 24066501 | Alfred Ogunbayo | ao634@kent.ac.uk |
| 25042127 | Priscilla Baiya | pb543@kent.ac.uk |
| 25040192 | Isabelle Bricaud | ib349@kent.ac.uk |
| 25041320 | Suchi Prommuangkun | sp991@kent.ac.uk |
| 25021661 | Napatchol Thaipanich | nt375@kent.ac.uk |

1. **Introduction**

Automated food classification and nutritional estimation have become increasingly vital for health monitoring, dietary tracking, and nutrition management. This project focuses on developing a machine-learning system that classifies food images and estimates their nutritional content. Using the Food-101 dataset, combined with data from the USDA’s Food Data Central, the system classifies food items and provides their nutritional values through a web interface.

Convolutional Neural Networks (CNNs) are employed due to their strong performance in image classification tasks. Additionally, transfer learning with models such as EfficientNet and InceptionV3 enhances the system's feature extraction capabilities, leading to improved classification accuracy. To benchmark the deep learning models, Random Forest and Decision Trees are also implemented, providing a comparative performance analysis.

This project aims to develop an AI-driven solution for food classification and nutritional estimation by integrating various machine learning techniques and nutritional databases, advancing health-related AI applications.

1. **Project Management and Development Flow**

**Requirements**

• **Dataset**: Food-101, USDA Food Data Central.

• **Tools:** Jupyter Notebook, Google Colab, Keras, TensorFlow, Scikit-learn, GitLab, PyCharm.

• **Note**: A GPU is recommended for training models with TensorFlow to accelerate processing.

**Project Management & Stages**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprints** | **Tasks** | **Milestones** | **Notes** |
| **Sprint 1**  (1 week) | Planning  Dataset Collection  Dataset Splitting Strategies.  Version Control | The dataset was successfully uploaded to GitLab.  Preprocessing tasks were completed on schedule.  Git Branches were created. | The dataset was loaded directly from Colab using TensorFlow datasets for better efficiency.  All code and model versions were managed via GitLab. |
| **Sprint 2**  (2 weeks) | Model Development  Iterative Experimentation on Baseline Models (Decision Trees) | Baseline models were trained, and the use of metrics for initial evaluation was completed. | Suboptimal performance with high-dimensional features.  Further experimentation with ensemble methods. |
| **Sprint 3**  (1 week) | Ensemble Methods Experimentation  (Random Forest)  Model Evaluation and Optimisation | Random Forests provided better results than decision trees.  Better Feature extraction and Hyperparameter tuning improved performance. | Random Forest showed early promise with performance, but it still relied on feature engineering, limiting its ability to extract complex patterns. |
| **Sprint 4**  (2 weeks) | Deep Learning and Transfer Learning:  CNN,  InceptionV3, and  EfficientNet  Model Evaluation | CNNs outperformed Decision Trees and Random Forest in classification accuracy.  EfficientNet achieved the highest accuracy while maintaining a balance between high precision and recall across the complete dataset. | Pre-trained models improved accuracy over traditional CNNs, but fine-tuning was computationally expensive.  InceptionV3 and EfficientNet  Performance comparison to guide towards model improvement. |
| **Sprint 5**  (1 week) | Model Extension  Final Model Selection  Real-Time Demo Testing | Integrated selected models by creating a JSON file with data from USDA’s Food Data Central to enable nutritional estimation for model extension.  Functional testing to evaluate detection accuracy was completed. | Successfully integrated food classification models with nutrition data and displayed the results on the web interface.  The initial testing was conducted using downloaded images in Jupyter Notebook, followed by user-uploaded images tested directly on the web interface. |
| **Sprint 6**  (1 week) | Report and documentation | Preparation of reports, statistics, performance comparison, presentation, and source code.  Design and References. | Report and Presentation finalised. |

*Table 1: Project Management & Stages of this Project*

**Development Flow**

Une image contenant texte, capture d’écran, Police, diagramme

Le contenu généré par l’IA peut être incorrect.

*Figure 1: Development Flow of this Project*

1. **Decision Tree**

This section explores the use of a Decision Tree with multiple configurations for the Food-101 dataset as a baseline approach to food image classification. Our goal was to evaluate a non-deep learning model for its interpretability, feature importance, and runtime performance combined with extracted features. Decision Trees are generally less suited to high-dimensional features compared to convolutional neural networks (Breiman et al., 1984), when paired with carefully engineered traditional computer vision image features, they help understand the importance of different feature extraction and selection techniques.

**Architectural Design**

Our model **Preprocessing** pipeline begins with resizing and normalising images to a fixed resolution, ensuring uniform input for the feature extraction pipeline, filtering the Food-101 dataset to the first 10 classes, and splitting it into an 80/20 training–validation ratio.

We explored several **Feature Extraction** methods, including.

* **Colour-based features:** Colour Histograms capture overall colour distribution
* **Texture Features:** Using Local Binary Patterns (LBP) and Gabor filters to characterise image texture.
* **Shape Features:** Histograms of Oriented Gradients (HOG) to capture orientations.
* **Deep Features:** Extracted from pre-trained CNNs EfficientNet (Less effective due to complex added noise).

The extracted features are concatenated into a high-dimensional vector. High vector dimensionality is managed with **Feature Selection Techniques** such as SelectKBest (using f\_classif), Recursive Feature Elimination (RFE), and Principal Component Analysis (PCA) for dimensionality reduction (Jolliffe, 2002).

The Decision Tree was implemented using Scikit-learn’s **DecisionTreeClassifier** and hyperparameters were optimised using GridSearchCV to balance models overfitting and underfitting. The training was executed on an A100 GPU in Google Colab, to efficiently handle large feature sets, and the **Evaluation Metrics** include Accuracy, F1-score, Confusion Matrices, Classification reports and Runtime performance metrics

**Implementation**

The implementation of our Decision Tree proceeds as follows:

1. **Data Filtering and Preprocessing:** Apply the standard preprocessing pipeline on all images, and normalise to each experiment's requirements (e.g., 64×64, 124 x 124).
2. **Feature Extraction**: Features are extracted from each image using the methods above. In some experiments, we merged basic traditional computer vision extracted features with deep features.
3. **Feature Selection:** We employed feature selection and dimensionality reduction techniques like; SelectKBest (f\_classif) to filter out weak features, RFE to further reduce the feature set, and PCA for high dimensionality of the raw extracted features (up to 8156 dimensions in some configurations).
4. **Model Training and Hyperparameter Tuning:** Train the decision tree with a balanced class weight, and GridSearchCV were used to test a range of hyperparameters like max\_depth, min\_samples\_split, and ccp\_alpha pruning to reduce overfitting, a common issue with Decision Trees when dealing with high-dimensional data.
5. **Model Evaluation:** The evaluation metrics were used to monitor performance progress and identify areas for further model hyperparameter tuning.

**Runtime Performance and GPU Configuration**

Training time varied significantly depending on the feature reduction ranging from 9 seconds with aggressive PCA (n\_components = 15), and when minimal dimensionality reduction like (n\_components = 200) is applied, training exceeded 10,000 seconds. Extensive feature extraction without proper reduction led to significantly longer training times. Due to the growth in tree complexity with increased feature dimensions, the use of an L4 and A100 GPU on Google Colab was essential to manage computational demands, slower GPUs failed due to memory constraints.

**Performance Evaluation**

The best-performing Decision Tree configurations achieved validation accuracies between 20–24%. This overall performance is poor compared to current deep learning approaches. The final model successfully balances training efficiency and generalisation to unseen examples while still requiring significant computational resources. Model 8 is a strong alternative if computational efficiency is a priority, though with slightly more overfitting.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Model** | **Features Extracted** | **Feature Selection** | **Validation / Training Accuracy** | **Training Time** |
| Final Model | HOG & Colour Histograms | PCA = 15 | Val: 0.225; Train: 0.257 | 988.16 sec |
| Test Model 9 | HOG & Colour Histograms | Kbest = 20; RFE = 5 | Val: 0.204; Train: 0.217 | 10371.12 sec |
| Test Model 8 | Local Binary Patterns (LBP), HOG, Gabor Filters, Colour Histograms. | Kbest = 100; RFE = 50 | Val: 0.225; Train: 0.281 | 368.65 sec |
| Test Model 7 | HOG, Colour Histograms & EfficientNet Output | PCA = 50 | Val: 0.214; Train: 0.248 | 2432.39 sec |

*Table 2: Performance of Decision Tree Experiment Variants on the First 10 Food Classes*

Experiments combining features extracted from the output of a pre-trained CNN EfficientNet merged with traditional computer vision features (Models 7 and 6) introduced noise and complexity, and the feature vector space became more complex, making it less interpretable for Decision Trees (Tan and Le, 2019). As a result, they had the highest training time among all models.

A chart of food items

AI-generated content may be incorrect.

*Figure 2: Confusion Matrix Plot (Final Model)*

**Encountered Challenges**

**Exponential Growth of Tree Complexity:** Without good feature selection techniques, decision trees grow exponentially in training time and tree complexity, especially when handling lots of extracted features.

**Balancing Overfitting vs. Underfitting:** Hyperparameter tuning became increasingly difficult as the combined feature vector size grew.

**Computational Demands:** High-dimensional feature sets required significant GPU resources; slower GPUs were inadequate.

While Decision Trees did not reach the performance levels of other models because they struggled to capture the complex relationships in high-dimensional image data, they provided valuable insights into feature importance and the effects of various feature extraction techniques. HOG and Colour Histograms with PCA outperformed CNN-enhanced versions in validation accuracy while maintaining significantly lower computational costs. The reduced feature space allowed for more efficient decision boundaries, preventing overfitting and improving generalisation. Future exploration for development includes ensemble methods such as Random Forests to enhance performance and tackle the challenges posed by high-dimensional feature representations.

1. **Random Forest**

We explored the use of Random Forest for image classification and optimised its performance using Grid Search with Cross-Validation (GridSearchCV). The hyperparameters tested included n\_estimators (50, 100, 200), max\_depth (10, 20, None), min\_samples\_split (2, 5, 10), and min\_samples\_leaf (1, 2, 4). We also tested different feature selection strategies (max\_features: None, 'sqrt', 'log2') and evaluated bootstrap sampling (True/False) (Scikit-learn, n.d.).

**Architectural Design**

To evaluate the effectiveness of Random Forest, we explored three feature extraction methods: Raw Pixel Training, where the model was trained directly on pixel values; Manual Feature Extraction, where features were extracted based on colour, texture, and shape; and Feature Extraction with EfficientNet, where a pre-trained deep learning model was used for feature embedding.

**Implementation**

**Raw Pixel Training**

For this approach, the images from the first 10 food classes were resized to 64×64 pixels, flattened into one-dimensional vectors, and normalized to the range [0,1]. The processed images were stored as NumPy arrays. The optimized Random Forest classifier was then trained on these raw pixel features, achieving an accuracy of 0.3353.

**Manual feature extraction**

To improve performance, we manually extracted Colour, texture, and shape features. Colour features were obtained by converting images from RGB to HSV to improve segmentation and robustness to lighting variations. Texture features were extracted using Gray-Level Co-occurrence Matrix (GLCM) to capture spatial relationships, while shape features were derived using Histogram of Oriented Gradients (HOG). Since HOG is independent of Colour, grayscale images were used to reduce computational costs.

After extracting features, we trained the Random Forest classifier separately on each feature set, as well as on combinations of features. The best result was obtained using a combination of Colour and texture features, achieving an accuracy score of 0.31, lower than the raw pixel-based model (GeeksforGeeks, n.d.).

**Feature Extraction with EfficientNet**

Manual feature extraction did not yield satisfactory results, prompting us to leverage the capabilities of Convolutional Neural Networks (CNNs) for feature extraction. We used a pre-trained EfficientNet model to extract deep features from the images. These high-level features, which capture complex patterns in the data, were then used to train the Random Forest classifier. This approach significantly improved performance, achieving an accuracy score of 0.7647—substantially higher than the results obtained from using Random Forest alone or with manually extracted features (Keras, n.d.).

**Evaluation**

Each of the 3 approaches was evaluated using a classification report which captures precision, f1-score and recall along with a confusion matrix. Table 2 compares the classification reports of the Random Forest model trained on EfficientNet features vs. the Random Forest model trained on the extracted features, while Figures 3 show their respective confusion matrices.

**Classification Report**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Random Forest with Feature Extraction** | | | **Random Forest with EfficientNet** | | |
| Food category | Precision | Recall | F1-score | Precision | Recall | F1-score |
| Beignets | 0.51 | 0.49 | 0.50 | 0.88 | 0.87 | 0.87 |
| Bibimbap | 0.42 | 0.56 | 0.48 | 0.91 | 0.87 | 0.89 |
| Apple\_pie | 0.23 | 0.17 | 0.19 | 0.62 | 0.50 | 0.55 |
| Bread\_pudding | 0.21 | 0.20 | 0.21 | 0.56 | 0.70 | 0.62 |

*Table 3: Classification reports*

Beignets and Bibimbap had the highest F1 scores (0.50 and 0.48) using Random Forest with manual feature extraction, but EfficientNet significantly improved to 0.87 and 0.89. Misclassified classes like Beet Salad, Apple Pie, and Bread Pudding (F1 scores of 0.19–0.21) also saw notable gains, with Bread Pudding rising to 0.62. Overall, the Random Forest model struggled—likely due to visual similarities between food items and its limitations in image classification tasks.

**Confusion Matrix**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

*Figure 3: Confusion Matrix*

**Runtime Performance**

-          Hyperparameter tuning with GridSearchCV; 3,468 seconds

-          Random Forest training: 47.69 seconds

**Encountered issues**

One major challenge encountered was that Random Forest is not well-suited for image classification, leading to low accuracy when trained on raw pixels or manually extracted features. The classifier struggled with high-dimensional image data, and computational constraints further limited experimentation.

1. **CNN**

A Convolutional Neural Network (CNN) is a Deep Learning technique that is particularly effective in the field of computer vision, especially for processing structured data such as images. It mimics the functioning of the human visual cortex by using convolutional layers to extract features from images, such as textures, shapes, contours, and colours. The feature extraction process is, therefore, automatic. Applying a CNN for food classification is a wise choice due to its ability to capture and interpret complex visual features, translational invariance, and performance, which has already been proven in similar tasks.

**Architectural Design**

The constructed CNN model follows a sequential approach. First, the model takes input images of size 224x224 pixels with 3 colour channels for RGB, as the images are in colour. The model is primarily built with convolutional blocks that integrate convolutional layers, where the filter size increases as the model progresses (32, 64, 128, 256), while the kernel sizes, conversely, become smaller (5x5, 3x3, 2x2). These layers enable the model to recognise patterns in the images.

Each Conv2D layer includes a ReLU activation function to incorporate non-linear properties, followed by a batch normalisation layer to stabilise training.

Between the convolutional blocks, pooling layers are interspersed with a pool size of 2x2 to perform downsampling and progressively reduce spatial dimensions, allowing the model to focus only on the most relevant features. Additionally, Dropout layers with a rate of 0.2 are added to aid regularisation and limit overfitting.

Finally, the model includes two dense layers: the first has 512 units, and the second has 101. The latter uses the softmax activation function to produce a probability distribution over the classes of the Food-101 dataset, allowing the input image to be classified into one of the food categories.

A close-up of a computer screen

AI-generated content may be incorrect.

*Figure 4: Architectural Design of the CNN implemented*

**Implementation**

To implement the model, we use TensorFlow and Keras. First, we ensure that the dataset is loaded using TensorFlow Datasets (TFDS). For image preprocessing, we resize all images to 224x224 and normalise them between 0 and 1.

We then define our model as described earlier and compile it using the *Adam* optimiser. Adam combines the advantages of two optimisation methods: AdaGrad (Adaptive Gradient Algorithm) and RMSProp (Root Mean Square Propagation). Instead of adapting the learning rates of parameters based solely on the first moment estimate, Adam also considers the second moment estimate of the gradients.

During compilation, *Sparse Categorical Crossentropy* is used as the loss function. This is beneficial because it does not require labels to be one-hot encoded, calculates loss efficiently, provides stable gradients, and is widely used for multi-class classification problems.

We implement *Mixed Precision Training* to improve performance and reduce memory usage. Finally, we train the model for 60 epochs with a batch size of 64, using callbacks such as *ReduceLROnPlateau* to dynamically adjust the learning rate based on validation loss.

**Runtime Performance and GPU Configuration**

The training is conducted on the Google Colab platform using the *entire dataset*. The GPU used is the A100 GPU. The runtime for each epoch is approximately 26 seconds, with a total training time of around *25 minutes and 49 seconds* for 60 epochs.

**Evaluation**

Metrics such as Accuracy, Loss, F1-score, Precision, Recall, Confusion matrix, and plots of accuracy and loss over the epochs during training and validation are used to assess the model's performance.

A diagram of a graph

AI-generated content may be incorrect.A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

AI-generated content may be incorrect.

|  |  |  |
| --- | --- | --- |
|  | **Accuracy** | **Loss** |
| Training | 0.85 | 0.50 |
| Validation | 0.66 | 1.56 |

*Figure 5: Final results of the simple CNN*

The model achieves a test accuracy of *65.94%* and an average macro F1-score of *0.6590*. This is due to some classes having very high F1-scores, while others perform significantly worse. The gap between training and evaluation suggests a possible overfitting issue. Additionally, the stagnation of accuracy from epoch 40 onward indicates that the model has reached its performance limit.

Some classes have very low precision, indicating that they are often misclassified as other classes. This is confirmed by the confusion matrix visualisation, where certain classes are poorly recognized, even though the overall diagonal pattern is still visible.

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** |
| Apple\_pie | 0.38 | 0.34 | 0.36 |
| Ravioli | 0.41 | 0.42 | 0.41 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** |
| Edamame | 0.98 | 0.96 | 0.97 |
| Macarons | 0.90 | 0.80 | 0.85 |

*Table 5: Top2 & Bottom2 of the simple CNN*

By analysing the best and worst recognised classes from the classification report, we can conclude that foods with distinct characteristics are well classified, whereas those with similar features—such as differentiating between two types of meat—are more challenging to distinguish.

Since the results are limited with a simple CNN, we implemented *pre-trained models* likeInceptionV3 and EfficientNet, which should significantly improve performance.

1. **InceptionV3**

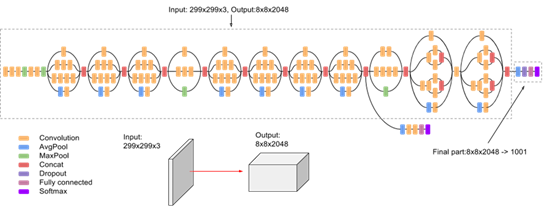
InceptionV3 is a deep convolutional neural network (CNN) architecture designed for efficient and accurate image classification. Developed by Szegedy et al. (2016), it introduces factorised convolutions, auxiliary classifiers, and batch normalisation, improving computational efficiency while maintaining high accuracy. The model uses inception modules, which apply multiple convolutional filters in parallel to capture diverse spatial features and a Global Average Pooling (GAP) layer to reduce dimensionality. With an input size of 299 × 299 × 3, InceptionV3 is well-suited for complex image classification tasks such as food recognition due to its ability to capture multi-scale spatial information.

**Architectural design**

Unlike traditional CNN architectures that stack sequential convolutional layers, InceptionV3 divides feature extraction into multiple parallel paths using 1×1, 3×3, and 5×5 convolutions. To enhance computational efficiency, larger convolutions (e.g., 5×5) are factorized into smaller operations, such as 1×3 followed by 3×1, reducing the number of parameters while maintaining feature extraction capabilities (Szegedy et al., 2016).

A key innovation in InceptionV3 is the extensive use of batch normalization across the network, which stabilizes gradient updates and improves convergence speed (Ioffe & Szegedy, 2015). Additionally, auxiliary classifiers are placed at intermediate layers during training. These act as secondary softmax classifiers, helping to propagate gradients deeper into the network and mitigate the vanishing gradient problem, thereby accelerating convergence.

Rather than employing traditional fully connected layers, InceptionV3 incorporates Global Average Pooling (GAP) at the final stage, reducing the number of parameters and enhancing model generalization. This approach minimizes overfitting by averaging feature maps rather than relying on fully connected layers, which are prone to overfitting in deep networks. The final classification layer employs a softmax activation function, producing probability distributions over 101 food categories in the Food-101 dataset. The figure below illustrates the InceptionV3 architecture, highlighting its modular design, parallel convolutional filters, and auxiliary classifiers, all of which contribute to improved training efficiency and convergence.



*Figure 6: InceptionV3 architecture (Xie et al., 2019)*

**Implementation**

The Food-101 dataset was resized to 299×299×3 to match InceptionV3’s input requirements and normalised using InceptionV3’s preprocessing function. Images were processed in parallel with TensorFlow’s API for efficient loading. Data augmentation techniques, including random horizontal flipping, contrast adjustment, random cropping, brightness modifications, and zoom, were applied to enhance model generalization. The augmented training dataset was batched and preloaded for efficient training.

The InceptionV3 model was initialised with pre-trained ImageNet weights, with the classification head removed and replaced by a custom head consisting of a Global Average Pooling (GAP) layer, a 256-unit dense layer with ReLU activation, and a final softmax output layer for the 101 food categories. Dropout (0.5 probability) was applied to prevent overfitting. The base model was frozen initially to retain ImageNet features. The Adam optimiser with a learning rate of 0.0001 was used, and the model was trained with early stopping and a scheduler to adjust the learning rate based on validation loss.

After initial training, fine-tuning was performed by unfreezing the last layers of the base InceptionV3 model while keeping the first 200 layers frozen. The learning rate was reduced to 0.00001 to ensure stable updates. Fine-tuning continued to improve accuracy and reduce overfitting. The final trained model was saved for future use.

**Runtime Performance and GPU Configuration**

The model was trained on Google Colab with an A100 GPU, ensuring efficient computation. The training process was divided into two phases. The initial training phase consisting of 50 epochs, took 2 hours and 52 minutes. The fine-tuning phase included 25 epochs with early stopping required 1 hour and 27 minutes. The total training time is approximately 4 hours and 20 minutes.

**Evaluation**

The classification report highlights varying performance across food classes. Precision and recall ranged from 0.51 to 0.87, with strong performance for visually distinct items like *baklava* (precision: 0.81, recall: 0.86) and beef\_*carpaccio* (precision: 0.81, recall: 0.80). In contrast, *apple\_pie* exhibited lower recall (0.51), suggesting misclassification with similar-looking foods. Overall, the model achieved 76.63% test accuracy, with a macro-averaged F1-score of 0.7663, indicating strong classification performance across all categories. The test loss (0.93) reflects a well-generalised model, with fine-tuning significantly reducing test loss by approximately 36%, further improving performance.

**A graph of different colored lines

AI-generated content may be incorrect.A screen shot of a graph

AI-generated content may be incorrect.**

*Figure 7: InceptionV3 Plots for Accuracy and Loss, and Confusion Matrix*

**Encountered issues**

During training, overfitting was initially observed, particularly for food items with high contrast. This was mitigated by increasing dropout to 0.5. Additionally, class imbalance in the dataset resulted in biased predictions toward more frequently occurring food categories. To address this, class weighting was applied during loss computation. Another challenge was slow convergence, which was improved using a cosine decay learning rate schedule to adapt learning rates dynamically. Memory constraints when handling high-resolution images were resolved through gradient checkpointing, reducing GPU memory usage without compromising performance.

1. **EfficientNet**

EfficientNet is a deep learning model pre-trained on ImageNet. It is widely used for image classification tasks such as prediction and feature extraction. It requires input normalisation to the [-1, 1] range (Keras Team, n.d.). The original EfficientNet (B0–B7) uses compound scaling (BytePlus Editorial Team, 2025), while EfficientNetV2 (B0–B3, S, M, L) enhances performance with fused-MBConv layers via Neural Architecture Search (Song, 2022).

EfficientNetV2-B2 achieved the highest accuracy on the Food-101 dataset compared to the subversion of EfficientNet. This section discusses its architecture, implementation, runtime and GPU setup, evaluation, and challenges.

**Architectural Design**

The pre-trained EfficientNetV2-B2 model, with a total of 142 layers, is structured into eight main parts: a Stem Block, six convolutional blocks, and fully connected output layers. It processes 260×260 RGB images (Fu, 2023), starting with normalisation and a 3×3 convolution. Blocks 1–3 use Fused-MBConv layers without SE for speed, while Blocks 4–6 apply MBConv with SE to enhance feature extraction. The final layers include Global Average Pooling, Batch Normalisation, ReLU activation, Residual Connections, and Dropout to prevent overfitting, ending with Dense layers and Softmax for classification.

A diagram of a multicolored rectangular object

AI-generated content may be incorrect.A diagram of a diagram

AI-generated content may be incorrect.

*Figure 8: Architectural Design of EfficientNetV2B2 with* *MBConv and Fused-MBConv (Tan and Le, 2021)*

**Implementation**

The final model was built using TensorFlow and Keras, with data loaded via TFDS. Input images were resized to 260×260 and normalised, with data augmentation (e.g., flipping, rotation, brightness/contrast changes) applied to improve accuracy. The EfficientNetV2-B2 base was used without its top layer, followed by Global Average Pooling, Dropout, and a Dense layer with 101 Softmax outputs. The model was trained using the Adam optimiser (learning rate 0.0001), sparse categorical cross-entropy, and a batch size of 128. Early Stopping and a Learning Rate Scheduler were used, and during fine-tuning, all layers were made trainable to improve performance.

**Runtime Performance and GPU Configuration**

The training is conducted on the Google Colab platform using the entire dataset. The GPU used is the A100 GPU. The runtime for each epoch is approximately 100 seconds for the Initial stage, the frozen weight stage, which had 100 epochs and took about 2.82 hours, and approximately 110 seconds for the fine-tuning stage, which had 54 epochs and took about 1.71 hours. The entire process took around 4.53 hours.

**Evaluation**

This section will cover plots of accuracy and loss over the epochs during training and validation, showing the model's performance and the accuracy, loss, F1-score, precision, recall, and confusion matrix.

|  |  |  |
| --- | --- | --- |
|  | **Accuracy** | **Loss** |
| **Training** | 0.7676 | 0.8502 |
| **Validation** | 0.7676 | 0.8915 |
| **Prediction** | 0.8493 |  |

Une image contenant texte, ligne, Tracé, diagramme

Le contenu généré par l’IA peut être incorrect.

*Table 6: Accuracy & Loss in the Table and Line Chart Figure 9: Accuracy & Loss in the Table and Line Chart*

The EfficientNetV2-B2 model shows strong and balanced performance across training, validation, and prediction. After fine-tuning, both training and validation accuracies reached **0.7676**, with closely aligned losses, indicating no overfitting or underfitting. Accuracy and loss curves confirm steady learning and generalization. The model achieved a higher prediction accuracy of **0.8493** on unseen data, highlighting its robustness and effectiveness after fine-tuning.

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** |
| Edamame | 0.99 | 0.98 | 0.99 |
| Macarons | 0.97 | 0.97 | 0.97 |

A graph of confusion matrix

AI-generated content may be incorrect.

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** |
| Stack | 0.57 | 0. 52 | 0. 54 |
| Pork\_chop | 0.67 | 0. 57 | 0. 63 |

*Figure 10: The lowest and highest accuracy of classes and the confusion matrix*

The classification results show that visually distinct classes are easier to identify, while similar ones, like *steak* and *pork\_chop,* are often confused. *Steak* had the lowest F1-score (**0.542**), while *edamame* (**0.990**) and *macarons* (**0.972**) had the highest. The confusion matrix showed a Strong Diagonal Line, which confirmed these patterns, with clear diagonals for well-classified classes and scattered errors for harder ones.

**Encountered challenges**

1. High computational cost: Only EfficientNetB0 with 10 classes worked with a Jupyter Notebook on Raptor, but other models required substantial GPU resources.
2. Fine-tuning sensitivity: Incorrect learning rate adjustments resulted in overfitting or slow convergence.
3. Misclassification of similar foods: Certain foods with similar textures or colours were frequently confused.
4. **Comparison**

The comparison of models is carried out **only on the first 10 classes of the dataset** for ease.

A screenshot of a computer

AI-generated content may be incorrect.Une image contenant texte, capture d’écran, nombre, Police

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, menu, Police

Le contenu généré par l’IA peut être incorrect.

*Figure 11: Classification Reports : Simple CNN - RandomForest - DecisionTree*

A screenshot of a computer screen

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.

*Figure 12: Classification Report for InceptionV3 - EfficientNetV2B2*

By analysing the classification reports of each model, we first notice that for Random Forest and Decision Tree, the number of supports varies across the classes, while for the CNNs, it remains constant (250). This indicates an underrepresentation of some classes compared to others, which can impact the model's performance. Therefore, it is essential to have a balanced dataset.

Next, by examining the average scores of each model, we observe that Decision Tree achieves around 0.22, Random Forest around 0.32, the simple CNN around 0.72, InceptionV3 around 0.87, and EfficientNetV2B2 around **0.92**. Overall, EfficientNetV2B2 stands out with excellent performance, closely followed by InceptionV3.

By examining each classification report in detail, it is possible to identify the best-ranked classes as well as those that are ranked lower.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Top Class** | | | | **Bottom Class** | | | |
| **Class** | **Precision** | **Recall** | **F1-Score** | **Class** | **Precision** | **Recall** | **F1-Score** |
| **Decision Tree** | | | | | | | |
| Beet\_salad | 0.40 | 0.24 | 0.30 | Beef\_tartare | 0.00 | 0.00 | 0.00 |
| Bibimbap | 0.35 | 0.40 | 0.37 | Bread\_pudding | 0.12 | 0.12 | 0.12 |
| **EfficientNetV2B2** | | | | | | | |
| Baby\_back\_ribs | 0.96 | 0.97 | 0.97 | Apple\_pie | 0.84 | 0.76 | 0.80 |
| Bibimbap | 0.96 | 0.98 | 0.97 | Bread\_pudding | 0.79 | 0.87 | 0.83 |
| **Simple CNN** | | | | | | | |
| Bibimbap | 0.83 | 0.87 | 0.85 | Apple\_pie | 0.47 | 0.36 | 0.41 |
| Beef\_carpaccio | 0.89 | 0.86 | 0.87 | Bread\_pudding | 0.54 | 0.64 | 0.59 |
| **RandomForest** | | | | | | | |
| Beignets | 0.47 | 0.51 | 0.49 | Apple\_pie | 0.14 | 0.13 | 0.13 |
| Bibimbap | 0.46 | 0.57 | 0.51 | Breakfast\_burrito | 0.21 | 0.15 | 0.18 |
| **InceptionV3** | | | | | | | |
| Beignets | 0.92 | 0.95 | 0.94 | Apple\_pie | 0.79 | 0.68 | 0.73 |
| Bibimbap | 0.96 | 0.96 | 0.96 | Bread\_pudding | 0.75 | 0.82 | 0.78 |

*Table 7: Top 2 & Bottom 2 for Decision Tree, EfficientNetV2B2, A simple CNN, Random Forest, and InceptionV3*

Among all the models, a pattern emerges: each *performs well overall on the same classes* (bibimbap consistently appears in the top 2) and *faces difficulties with the same others* (*apple\_pie* and *bread\_pudding* frequently appear in the bottom 2). This suggests that bibimbap has more distinctive features, while *apple\_pie* and *bread\_pudding* have traits that are harder to differentiate or too similar to other classes.

In all cases, InceptionV3 and EfficientNetV2B2 show the best performance. These results confirm that certain models, such as Random Forest or Decision Tree, are not suitable for computer vision tasks, while CNNs, especially those pre-trained on ImageNet, excel at image classification.

.A graph of a patient's function

AI-generated content may be incorrect.A graph of different colored lines

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.

*Figure 13: Plots of ROC-AUC: InceptionV3 - CNN - EfficientNetV2B2*

By comparing the three types of CNNs, the analysis of the ROC curves and AUC allows for a more detailed evaluation of their ability to distinguish the classes and their false positive rate. According to the plots, the curves of the simple CNN are less steep at the beginning, with some progressing slowly, indicating a higher false positive rate and less effective discrimination between certain classes.

In contrast, for the pre-trained models, the curves show a steeper rise, particularly for InceptionV3, whose curves are almost perfect. This indicates a very low false positive rate and excellent class separation. Based on these observations, InceptionV3 shows the best performance.

However, by examining all the metrics, EfficientNetV2B2 offers the best combination, considering both class separation and the overall balance of the model in terms of accuracy. This highlights the importance of fully understanding the problem at hand to choose the most suitable model.

Both EfficientNetV2B2 and InceptionV3 were run on the A100 GPU. Yet, their respective execution times are 4 hours and 31 minutes and 4 hours and 20 minutes. This suggests that if access to a powerful GPU is available and good results are sought, EfficientNetV2B2 remains the best choice.

In conclusion, **EfficientNetV2B2 proves to be the best model**, as its architecture maintains a balance between high precision and high recall, which is essential for classification problems like those in Food-101 dishes. Its overall performance, such as the average F1-score and AUC-ROC scores, is superior, even for difficult-to-classify classes like *apple\_pie* and *bread\_pudding*. It is the most efficient across the entire dataset, with high performance and good generalisation capabilities, along with satisfactory computational performance.

Models like Decision Tree and Random Forest do not perform well because they struggle to extract complex features from images and are not suited for capturing spatial data. This is why when EfficientNet is used for feature extraction with Random Forest, the performance improves immediately. Although the simple CNN shows decent performance, its architecture is less advanced, and it has more difficulty detecting features since it starts from scratch. In contrast, pre-trained models like InceptionV3 and EfficientNetV2B2 are optimised for this task. These two models outperform the others because they are specifically designed to capture image features with well-suited architectures while avoiding overfitting.

1. **Model Testing and Web Deployment**

After training a model, it is essential to ensure that it performs correctly on previously unseen images. To achieve this, we will test our models in real time using actual food photos captured directly by a camera.

For this, we use two methods. The first involves directly testing the models on Jupyter Notebook by downloading an image from a URL, applying the necessary preprocessing to ensure the input meets the model's expectations, and then making a prediction to display the class and the result.

The second method involves testing the model with images taken directly by the user. For this, it is essential to save the model's architecture and weights in an .h5 file, allowing it to be rebuilt, then applying the appropriate preprocessing and making predictions in the same way. The entire process is then integrated into a web interface, providing the option to select a real photo.

As part of the implementation of a selection of CNN models for predictions, we had to pay particular attention to image preprocessing before using them with a specific model. This turned out to be one of the biggest challenges we encountered. We initially assumed that image normalisation was uniform across all models, simply done by dividing by 255. However, TensorFlow's preprocessing function for EfficientNet seems to adopt a more complex approach. During our tests with real images using division by 255, we consistently got the same prediction. It was crucial to apply the same normalisation method during both training and testing. For this model, therefore, we decided to use TensorFlow's preprocessing function directly.

Finally, due to time constraints for implementing nutritional value estimation through Deep Learning methods based on image segmentation and automatic calorie calculation, we opted for an alternative approach. The integration of the web interface allows us to link the classes to a JSON file containing the nutritional values of the foods from the Food-101 dataset. These data were retrieved from the USDA Food Central dataset, enabling us to display basic nutritional information for an identified dish.  However, this method does not account for the actual nutritional quantity of the dish present in the image.

1. **Conclusion and Future Work**

This project demonstrated the effectiveness of CNNs and transfer learning for food classification, with InceptionV3 and EfficientNet improving accuracy by leveraging pre-trained knowledge. However, our testing showed that no single model consistently outperforms others, as CNNs trained from scratch can sometimes yield better results depending on the dataset. While Random Forest and Decision Trees provided a comparative baseline, they lacked the spatial feature extraction capabilities of deep learning models. Notably, combining Random Forest with EfficientNet resulted in better performance than using Random Forest alone, suggesting that hybrid approaches can enhance classification accuracy.

A key challenge remains accurate nutritional estimation, as portion sizes cannot be determined from images alone. Future work could integrate data segmentation to improve precision, as studies have shown that segmentation-based methods can enhance the accuracy of food weight and nutrient estimation (Min et al., 2024; Pouladzadeh et al., 2014). However, challenges such as variations in plating and viewing angles persist (Liu et al., 2023). Incorporating multi-modal learning or depth sensors could further improve real-world applicability by providing additional contextual information.

1. **Contributions and Code**

For the Comparison part, the files used are the following on Gitlab:

***DecisionTree***: 01\_FINAL\_MODEL\_DecTree\_HOG\_and\_Color\_Histogram\_PCA.ipynb

***RandomForest***: food101.ipynb under Raw Pixels section

***CNN***: CNN\_10\_classes\_final\_version\_ROC.ipynb

***InceptionV3***: *(Folder ver3)* inception\_v3\_food101\_10class.ipynb

***EfficientNetV2B2***:  colab.ipynb under EfficientNetV2B2 section

Each model was implemented by a group member, and the code was pushed to GitLab on the corresponding *branches* throughout the project. Once all tests were completed, everything was merged into the *main* branch.

All models and web code are available at the following link: [Isabelle Bricaud / AI System - Food Image Classification for Nutritional Estimation · GitLab](https://git.cs.kent.ac.uk/ib349/ai-system-food)

**References**

**Books:**

* Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984) *Classification and Regression Trees.* Monterey, CA: Wadsworth.
* Chollet, F. (2017) *Deep Learning with Python.* Manning Publications.
* Jolliffe, I.T. (2002) *Principal Component Analysis.* 2nd edn. New York: Springer.

**Journal Articles:**

* Bossard, L., Guillaumin, M. and Van Gool, L. (2014) ‘Food-101 – Mining Discriminative Components with Random Forests’, *European Conference on Computer Vision (ECCV)*, pp. 446–461. Available at: <https://data.vision.ee.ethz.ch/cvl/datasets_extra/food-101/static/bossard_eccv14_food-101.pdf> [Accessed 7 February 2025].
* Liu, Y., Wang, L. and Zhang, H. (2023) ‘A novel approach to estimate the weight of food items based on image analysis and boosting regression’, *Scientific Reports.* Available at: <https://www.nature.com/articles/s41598-023-47885-0> [Accessed 20 March 2025].
* Min, W., He, M., Ding, Y., Xu, J., Luo, Z. and Jiang, S. (2024) ‘Visual nutrition analysis: leveraging segmentation and regression’, *Frontiers in Nutrition.* Available at: <https://www.frontiersin.org/articles/10.3389/fnut.2024.1469878/full> [Accessed 22 March 2025].
* Pouladzadeh, P., Shirmohammadi, S. and Al-Maghrabi, R. (2014) ‘Estimating nutritional value from food images based on semantic segmentation’, *ACM International Conference on Multimedia.* Available at: <https://dl.acm.org/doi/10.1145/2638728.2641336> [Accessed 22 March 2025].

**Conference Papers:**

* Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2016) ‘Rethinking the Inception Architecture for Computer Vision’, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826.
* Tan, M. and Le, Q. (2019) ‘EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks’, *Proceedings of the 36th International Conference on Machine Learning*, pp. 6105–6114.

**Preprints:**

* Tan, M. and Le, Q.V. (2021) ‘EfficientNetV2: Smaller Models and Faster Training’, *arXiv preprint*, arXiv:2104.00298. Available at: <https://arxiv.org/pdf/2104.00298v3> [Accessed 19 March 2025].

**Web Articles and Documentation:**

* BytePlus Editorial Team (2025) ‘What is EfficientNet? Understanding CNN architecture’. Available at: <https://www.byteplus.com/en/topic/401740?title=what-is-efficientnet-understanding-cnn-architecture> [Accessed 19 March 2025].
* GeeksforGeeks (n.d.) ‘Random Forest for Image Classification using OpenCV’. Available at: <https://www.geeksforgeeks.org/random-forest-for-image-classification-using-opencv/> [Accessed 20 March 2025].
* HPE Africa (n.d.) ‘Qu’est-ce qu’un réseau neural convolutif (CNN)?’ Available at: <https://www.hpe.com/emea_africa/fr/what-is/convolutional-neural-network.html> [Accessed 22 March 2025].
* Keras Team (n.d.) ‘Keras Applications’. Available at: <https://keras.io/api/applications/> [Accessed 19 March 2025].
* Keras Team (n.d.) ‘EfficientNet B0 to B7’. Available at: <https://keras.io/api/applications/efficientnet/> [Accessed 19 March 2025].
* Keras Team (n.d.) ‘EfficientNetV2 B0 to B3 and S, M, L’. Available at: <https://keras.io/api/applications/efficientnet_v2/> [Accessed 19 March 2025].
* Keras (n.d.) ‘Image classification with EfficientNet and fine-tuning’. Available at: <https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/> [Accessed 24 March 2025].
* Scikit-learn (n.d.) ‘Random forests’. Available at: <https://scikit-learn.org/stable/modules/ensemble.html#forest> [Accessed 20 March 2025].
* TensorFlow Documentation (n.d.) ‘Transfer Learning and Fine-Tuning with Keras’. Available at: <https://www.tensorflow.org> [Accessed 20 March 2025].
* USDA FoodData Central (n.d.) ‘FoodData Central Food Search’. Available at: <https://fdc.nal.usda.gov/food-search> [Accessed 14 March 2025].
* Yixing Fu (2023) ‘Image classification via fine-tuning with EfficientNet’. Available at: <https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/> [Accessed 19 March 2025].

**Technical Reports & Theses:**

* Ciocca, G., Napoletano, P. and Schettini, R. (n.d.) ‘CNN-based Features for Retrieval and Classification of Food Images’. Available at: <https://boa.unimib.it/retrieve/e39773b8-e901-35a3-e053-3a05fe0aac26/CNN-based%20features_post-print.pdf> [Accessed 22 March 2025].
* Jiang, M. (2019) ‘Food Image Classification with Convolutional Neural Networks’. *Stanford University.* Available at: <https://cs230.stanford.edu/projects_fall_2019/reports/26233496.pdf> [Accessed 3 February 2025].
* Rudraja, V. (2022) ‘Food Image Classification Using Various CNN Models’, *International Journal of Innovative Research in Technology (IJIRT)*, 9(3), pp. 626-632. Available at: <https://ijirt.org/publishedpaper/IJIRT156433_PAPER.pdf> [Accessed 7 February 2025].