NATIONAL ECONOMICS UNIVERSITY

FACULTY OF MATHEMATICAL ECONOMICS

MAJOR ASSIGNMENT

Course: Deep Learning

Topic: Food Images Classification Using CNN and Transfer Learning

Instructor: Dr. Nguyen Thi Kim Ngan

Student information:

Full name	Student ID
Lê Thu Hằng	11222080
Võ Thị Minh Phương	11225327
Mai Trà Giang	11221769

Ha Noi, October 2025

Part 1: Theoretical Background of Convolutional Neural Networks (CNNs)

1. Introduction

Deep learning has transformed computer vision by replacing hand-crafted image processing with learned representations. Modern CNNs **automatically learn hierarchical features** from raw images. CNNs mirror the human visual pathway by building progressively complex feature hierarchies: early layers detect low-level patterns (edges, colors), and deeper layers combine these into high-level concepts. This end-to-end learning approach overcomes the limitations of classical methods and has led to breakthroughs in many AI applications. For example, CNNs now power autonomous driving (object/scene recognition) and medical imaging analysis, leveraging vast datasets and GPU compute to achieve human-competitive accuracy. They have also been applied to specialized tasks such as food image recognition: for instance, the Food-101 dataset (101,000 images across 101 categories) is widely used to benchmark CNN models on dish classification

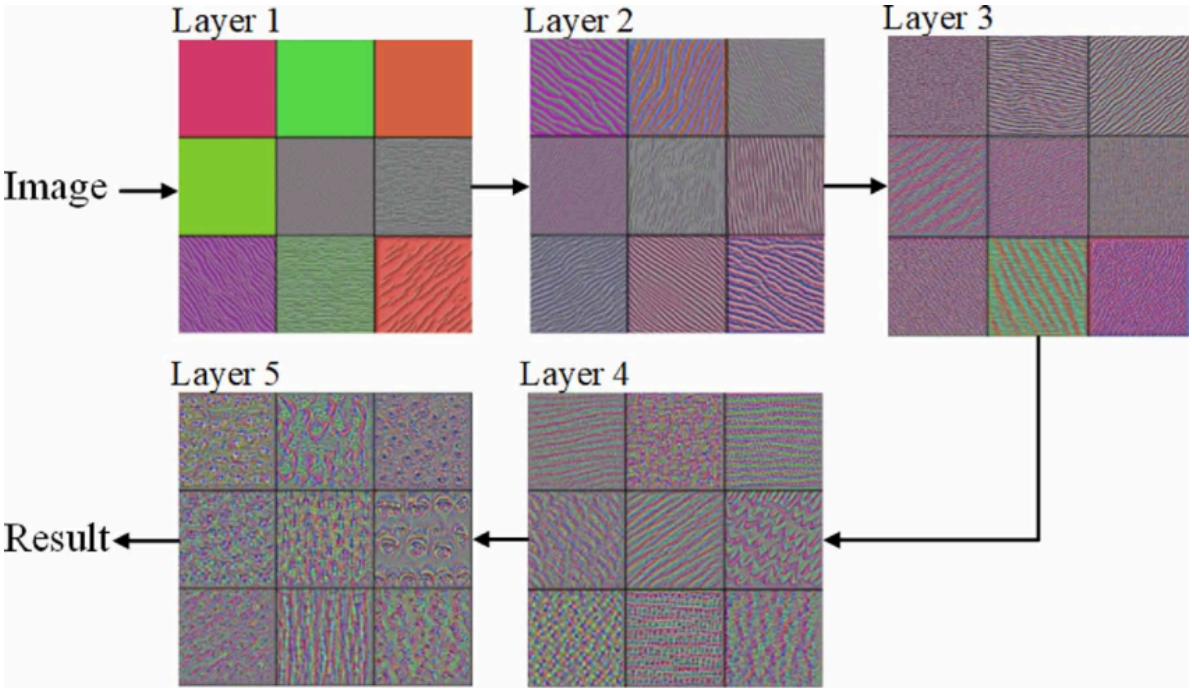

Figure 1: An illustration of the hierarchical feature of CNNs.
Source: [ResearchGate]

2. History of Convolutional Neural Networks

The inspiration for CNNs traces back to neuroscience. In 1959 Hubel and Wiesel discovered that neurons in the cat's visual cortex respond to specific edge orientations ("simple cells") and combinations thereof ("complex cells"), showing that the brain builds images from basic features. CNNs adopt a similar hierarchical, layered structure to process images. The first practical CNN was **LeNet-5** (LeCun et al., 1998), a 7-layer network for handwritten digit recognition. LeNet-5 combined convolutional layers with subsampling (pooling) and fully-connected layers to classify MNIST digits; it demonstrated that learned filters could outperform manual feature engineering. However, CNNs lay dormant until 2012, when **AlexNet** (Krizhevsky et al., 2012) achieved a breakthrough by winning the ImageNet competition. AlexNet introduced ReLU activations, dropout regularization, and GPU-accelerated training, enabling much deeper networks. This success ignited a cascade of new architectures:

- **VGGNet (2014):** Used many stacked 3×3 convolutions (16–19 layers) to increase depth, demonstrating that very deep networks can improve accuracy. Its simple, uniform architecture made it effective but computationally heavy.
- **GoogLeNet/Inception (2014):** Introduced **Inception modules** that perform parallel convolutions at multiple scales (e.g. 1×1, 3×3, 5×5) within each layer, concatenating their outputs. It used 1×1 convolutions as bottlenecks to reduce dimensions, achieving high efficiency.
- **ResNet (2015):** Added *residual (skip) connections* that let layers learn modifications to the identity mapping (output = input + f(input)), effectively enabling networks over 100 layers deep. These skip connections solved the vanishing-gradient problem in very deep nets.
- **DenseNet (2017):** Each layer connects to every later layer in a "dense block," promoting feature reuse. DenseNets alleviate gradient vanishing, strengthen feature propagation, and drastically reduce the number of parameters.

Overall, modern CNN history progressed from simple shallow nets (LeNet-5) to very deep, specialized architectures (VGG, ResNet, DenseNet) that systematically address overfitting and training difficulties.

3. Definition

A **Convolutional Neural Network (CNN)** is a deep learning model specifically designed for grid-like data (e.g. images). A CNN takes an input tensor

$X \in \mathbb{R}^{H \times W \times C}$ (height H, width W, channels C) and processes it through multiple layers to produce an output (such as class probabilities). Crucially, a CNN preserves the spatial structure of the data: instead of flattening the image into a 1D vector, it uses convolutional filters that slide over 2D patches of the input. These learnable filters automatically extract local features (edges, textures) from images, and successive layers build up increasingly abstract representations. In effect, CNNs learn the parameters of these filters and other layers through backpropagation, enabling end-to-end mapping from raw pixels to output labels.

Mathematically, each convolutional layer performs operations like

$$\text{feature\_map}(x, y) = f\Big(\sum_{i,j,k} X[x+i, y+j, k] \cdot W[i, j, k] + b\Big),$$

where W is a filter (kernel) and f is an activation function (e.g. ReLU). By stacking such layers (with pooling and non-linearities in between), a CNN implements a complex hierarchical function on the input image.

4. **Why Use CNN for Image Processing**

CNNs are specifically well-suited for image data because they exploit **local spatial patterns**. The convolution operation ties together nearby pixels, allowing the network to learn edge detectors and texture filters that are position-independent. In contrast to a fully-connected network that treats every pixel separately, CNNs **preserve 2D locality**. Moreover, CNNs dramatically reduce the need for manual feature engineering: they **automatically learn features** from raw pixels through training, discovering the relevant patterns (e.g. edges, corners, object parts) without hand-design. Weight sharing (using the same filter at every image location) greatly reduces the number of parameters compared to a dense MLP, making CNNs more efficient and easier to train.

Another key advantage is *translation invariance*: because the same filters scan across the image, CNNs can recognize an object regardless of where it appears. Pooling layers further help by downsampling feature maps, making the network less sensitive to small translations. In practice, CNNs achieve state-of-the-art results on vision tasks (image classification, object detection, segmentation) precisely because they capture spatial hierarchies of features and generalize well to variations in position, scale, and appearance.

5. Comparison with Traditional MLP / Handcrafted Features

Aspect	Traditional MLP	CNN (Convolutional Neural Network)
Input Handling	Requires flattening 2D image into 1D vector, destroying spatial relationships	Takes full 2D image as input, preserving spatial structure
Feature Extraction	Features must be hand-engineered or learned from scratch on flattened input	Automatically extracts hierarchical features via convolutional filters
Parameters	Huge number of weights (every input pixel connected to every neuron)	Weight sharing via convolutional filters reduces number of parameters, prevents overfitting
Performance on Images	Struggles with large images or complex patterns; requires very large networks	Scales efficiently due to local connectivity and invariance; outperforms MLPs empirically
Suitability	Generic, not specialized for images	Specialized for images, computationally and statistically efficient

Table 1: Comparison between CNNs and Traditional MLPs

Source: Author's own compilation.

These differences are summarized in Table 1. In short, CNNs are specialized for images, whereas MLPs treat images as flat vectors and lack both computational and statistical efficiency in vision tasks.

6. Core Components of CNN

6.1. Convolutional Layer

A **convolutional layer** applies a set of learnable filters (kernels) to the input to produce feature maps. Each filter is a small matrix (e.g. 3×3 or 5×5) that is convolved across the image. At each position, the filter performs an element-wise multiplication with the underlying image patch and sums the results (plus a bias) to produce one output pixel in the feature map. Different filters capture different local

patterns. For example, one filter might detect horizontal edges while another finds circular shapes. Key hyperparameters of a conv layer include the *filter size*, *stride* (step between filter applications), and *padding* (whether to zero-pad the image borders). The result of a convolutional layer is a set of feature maps – one per filter – that highlight the presence of learned features (edges, textures, etc.) at each spatial location.

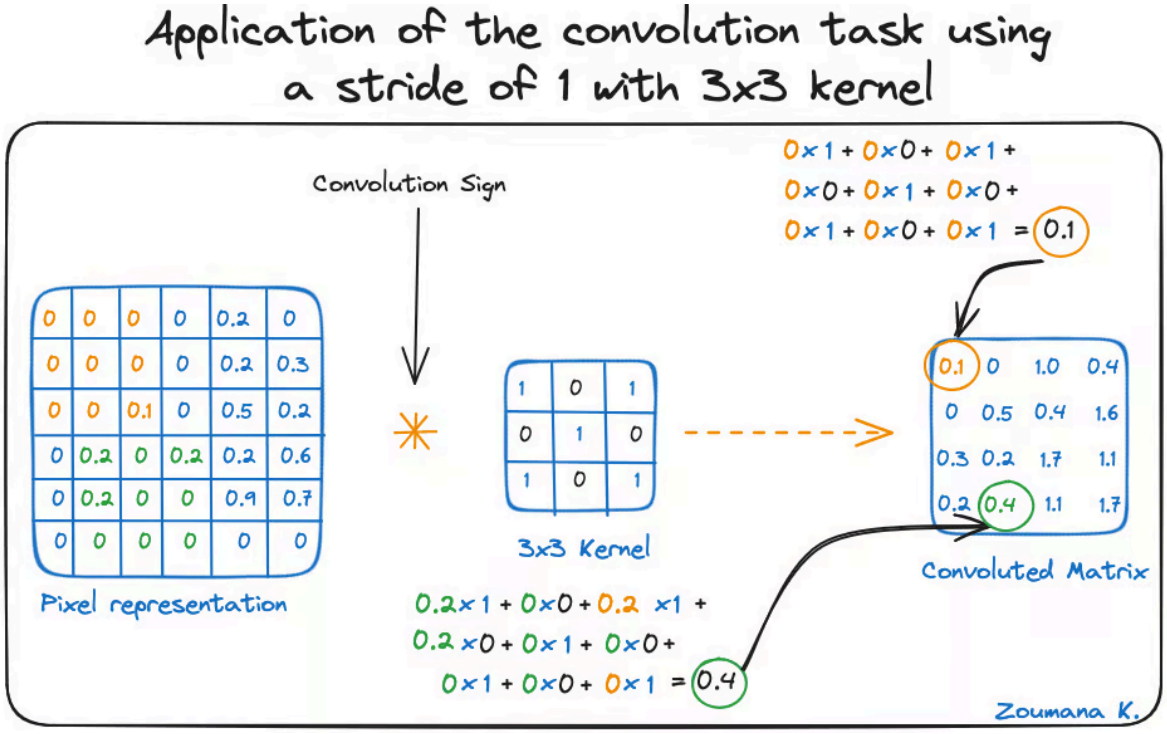

Figure 2: Application of the convolution task using a stride of 1 with 3x3 kernel

Source: [DataCamp]

6.2. Activation Functions

After each convolution, an **activation function** introduces non-linearity. The most common is the Rectified Linear Unit (ReLU), defined as f(x) = max (0,x). Applying ReLU after convolution helps the network learn non-linear relationships in the data and accelerates training by mitigating the vanishing-gradient problem. In practice, ReLU (or its variants like LeakyReLU) is used in every hidden layer. At the final layer for multi-class classification, a *Softmax* activation is typically used. The Softmax function transforms the network's raw output scores (logits) into class

probabilities that sum to 1. In this way, the network can be trained end-to-end using probabilistic interpretation.

6.3. Pooling Layers

A **pooling layer** reduces the spatial size of the feature maps to decrease computation and improve invariance. Commonly, *max pooling* is used: it partitions the feature map into non-overlapping regions (e.g. 2×2 windows) and outputs the maximum value from each region. This downsampling reduces the resolution of the features, lowering the number of parameters and computations in later layers. It also provides modest translation invariance by retaining only the strongest activations. Pooling combats overfitting by forcing the network to learn more robust, spatially invariant features. After pooling, the feature map has smaller width/height but the same number of channels. (Other pooling methods like average pooling exist, but max pooling is most common.) In summary, pooling *"pulls the most significant features"* from each patch, reduces feature dimensions, and thus acts as a form of dimensionality reduction and regularization.

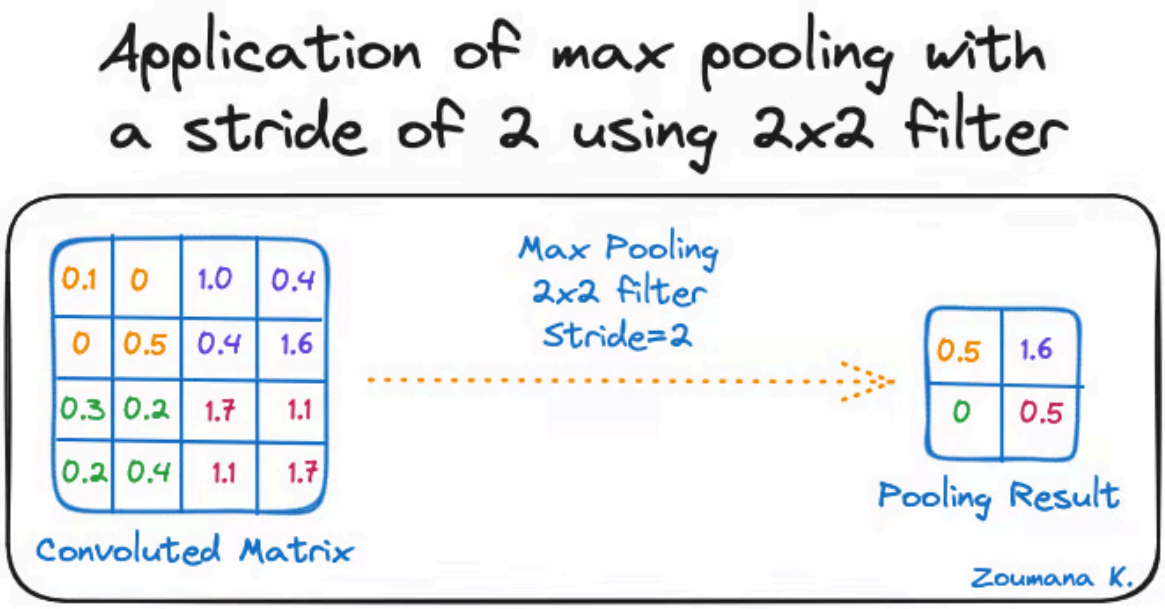

Figure 3: Application of max pooling with a stride of 2 using 2x2 filter

Source: DataCamp

6.4. Fully Connected Layers

Following convolutional and pooling stages, **fully connected (dense) layers** perform high-level reasoning. Before these layers, the final pooled feature maps are *flattened* into a 1D vector. This vector is then fed into one or more fully connected layers (standard neural network layers). In these layers, every input is connected to every output neuron, and typical activations (ReLU) are used. These layers learn complex, nonlinear combinations of the extracted features to perform classification or regression. Finally, for classification, the last layer is a Softmax that outputs a probability for each class. In practice (as in LeNet or AlexNet), one might have two or more dense layers (often large, e.g. 4096 units) before the final Softmax. These fully connected layers integrate the local features learned by earlier layers into a global decision.

6.5. Regularization Techniques

To prevent overfitting and improve generalization, several regularization techniques are widely used in CNNs:

(1) **Dropout:** During training, *dropout* randomly "drops out" (sets to zero) a fraction of neurons in a layer on each forward pass. This prevents neurons from co-adapting too strongly and effectively trains an implicit ensemble of subnetworks. Dropout is simple yet powerful: introduced by Srivastava et al. (2014), it significantly reduces overfitting in dense layers by forcing robustness. For example, a dropout rate of 0.5 means each neuron is dropped with 50% probability during training.

(2) **Batch Normalization:** BatchNorm normalizes the inputs of each layer to have zero mean and unit variance (based on the current mini-batch) and then scales/shifts them. Introduced by Ioffe and Szegedy (2015), it stabilizes and accelerates training by reducing *internal covariate shift*. By keeping layer inputs in a stable range, BatchNorm allows higher learning rates and often improves generalization (it has a mild regularizing effect). Many modern CNN architectures insert a BatchNorm layer after convolutions (and before ReLU) to speed up convergence.

(3) **Weight Regularization (L1/L2):** Adding a penalty on the magnitude of weights in the loss function (known as weight decay) constrains the model to prefer smaller weights. *L2 regularization* (squared weight penalty) is common: it discourages overly large weights and encourages the network to spread weight evenly across neurons. L1 regularization can produce sparse

features by driving some weights exactly to zero. Overall, weight regularization makes the learned model simpler and often more generalizable.

Each of these techniques – dropout, batch normalization, and weight decay – is often combined in CNNs to make training more robust and to reduce overfitting

7. Feature Propagation and Receptive Field

In a CNN, each neuron in a deeper layer has a larger *receptive field* on the input image. The **receptive field** of a neuron is the region of the original input image that can influence that neuron's output. Convolution and pooling layers limit each neuron's connections to a small local region of the previous layer, preserving spatial locality. However, as signals propagate through successive layers, these local regions "cascade" to cover larger patches of the input. Thus deeper neurons aggregate information from a wider context.

For example, the first convolutional layer might use 3×3 filters (receptive field = 3×3 pixels). Two layers later, each neuron's receptive field might be 5×5 or more on the original image (depending on stride/padding). This growth continues with depth and pooling. By the final layers, a neuron's receptive field can span almost the entire input, allowing high-level neurons to "see" whole objects. Intuitively, CNNs learn features at multiple scales: early layers see fine details, while deep layers see global structure. A larger receptive field helps capture long-range patterns (e.g. object composition). In dense prediction tasks like segmentation, ensuring a sufficiently large receptive field is crucial so that each output prediction uses enough of the input context (e.g. entire object).

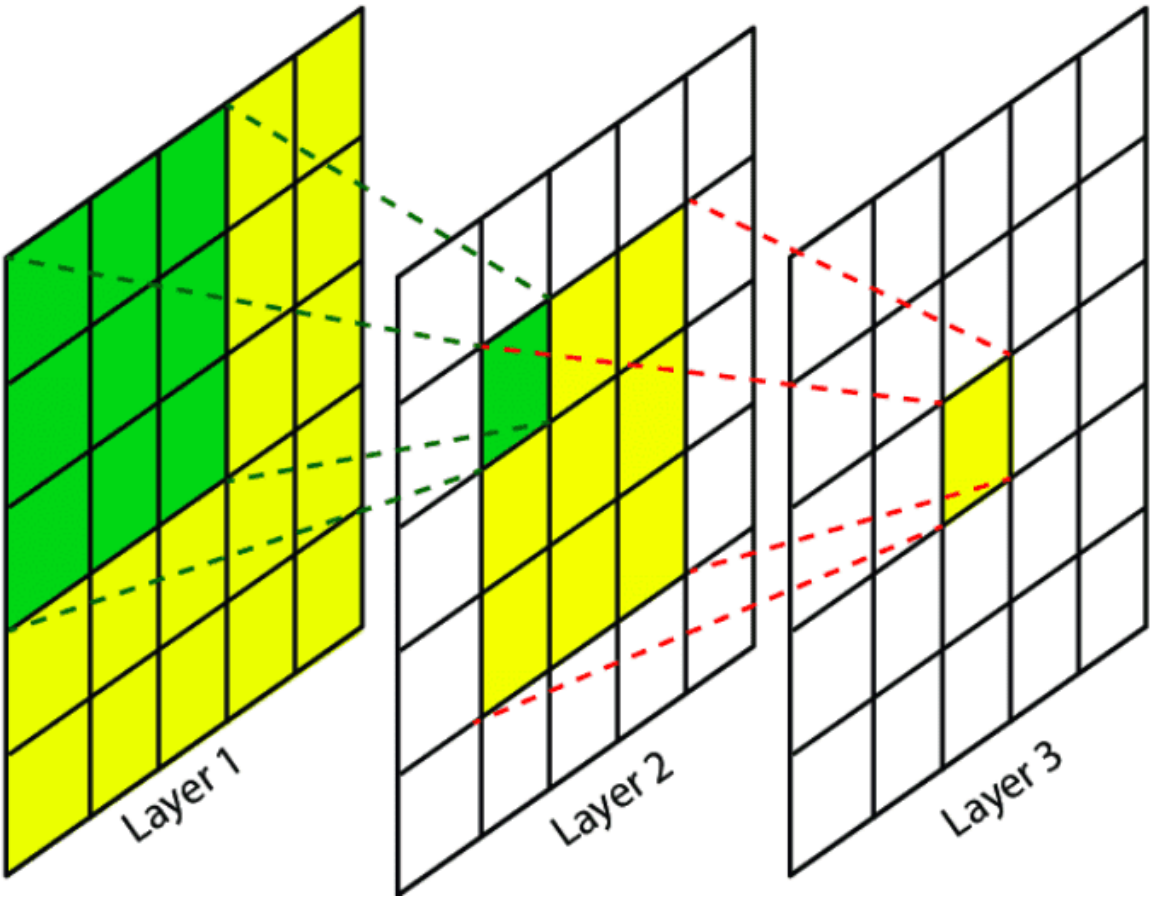

Figure 4: Growth of the receptive field in successive CNN layers

Source: ResearchGate

8. How CNN Learns Hierarchical Features

CNNs build up *hierarchical feature representations* layer by layer. As summarized in CNN textbooks and notes, the layers naturally specialize:

- **Early layers:** Detect very simple features such as edges, lines, and color blobs. These correspond to basic visual elements.
- **Mid-level layers:** Combine those edges into textures, patterns, and parts of objects (corners, shapes, stylized strokes).
- **Deep (high-level) layers:** Assemble mid-level features into semantic concepts like object parts or whole objects (e.g. faces, wheels) and even scene descriptors.

In effect, each layer "remembers" more abstract information. Fiveable's deep learning notes explain that low-level layers capture basic visual cues, intermediate layers form shapes/object parts, and high-level layers assemble complete object structures. This mirrors the visual cortex: simple cells feed into complex cells. In practice, one can visualize this hierarchy by looking at the feature maps: early-feature maps highlight edges in the image, whereas late-feature maps activate for entire objects. This progression from simple to complex is a key reason CNNs excel at vision tasks – they learn the appropriate features at each level of abstraction without human intervention.

9. Popular CNN Architectures

- **LeNet-5 (1998):** Yann LeCun's LeNet-5 was one of the first successful CNNs. It had 7 layers (convolutions, pooling, and fully connected layers) and was trained on the MNIST digit dataset. LeNet-5 demonstrated that convolutional and subsampling layers could be effectively combined to recognize handwritten digits. Its simplicity (few layers and parameters) makes it conceptually important but too small for modern large-scale tasks.
- **AlexNet (2012):** AlexNet was a milestone CNN that won ImageNet. Key ideas included using **ReLU activations** (which sped up training and mitigated vanishing gradients) and **dropout** regularization (to combat overfitting). Crucially, AlexNet was trained on GPUs, allowing a deep 8-layer network (5 conv + 3 FC). Its success proved that very deep CNNs could outperform traditional methods on large-scale images. (A limitation was its risk of overfitting – mitigated partly by dropout and data augmentation.)
- **VGGNet (2014):** VGG introduced the design principle of *very small convolution filters (3×3)* and very deep stacks. VGG-16 and VGG-19 had 16–19 layers with a repeating pattern of 3×3 conv blocks followed by max-pooling. This uniform architecture showed that depth alone, even with simple 3×3 filters, yields powerful models. Strengths: conceptually simple and strong accuracy. Drawbacks: extremely large number of parameters and high computational cost, due to many layers and large fully-connected layers.
- **GoogLeNet / Inception (2014):** GoogLeNet's Inception modules run multiple convolutions in parallel. Each Inception block applies 1×1, 3×3, and 5×5 filters (plus 1×1 before the larger ones for dimensionality reduction), concatenating their outputs. This *multi-scale feature extraction* allows the

network to capture patterns at different scales simultaneously. Another innovation was replacing heavy fully-connected layers with a global average pooling at the end. Strength: very efficient (fewer parameters than VGG) while still deep (22 layers). Limitation: complex design with many types of layers makes it hard to implement and tune.

- **ResNet (2015):** ResNet introduced **residual connections**. In a residual block, the input x is added to the output of a few stacked layers. This identity skip connection effectively allows gradients to flow directly through many layers. ResNets showed that networks could be trained with over 100 layers without accuracy degradation. Benefit: solves the vanishing gradient problem, enabling extremely deep nets. Drawback: the very deep models can be expensive to train and may overfit if not enough data is available.
- **DenseNet (2017):** In DenseNet each layer receives as input the feature-maps of *all* preceding layers. In other words, there are dense connections between layers. This design substantially strengthens feature propagation and reuse: gradients and information flow more easily, and every layer can access earlier features. DenseNets often achieve high accuracy with fewer parameters than equivalent networks. Strength: excellent gradient flow and parameter efficiency. Limitation: at training time the many concatenated feature maps consume more memory, making very deep DenseNets memory-intensive.

These architectures illustrate the evolution of CNN design: from simple (LeNet) to very deep, modular, and highly optimized models (ResNet, DenseNet). Each innovation addressed a new challenge (training speed, overfitting, gradient flow, computational efficiency) and pushed image recognition forward.

10. Optimization Algorithms and Loss Functions

CNN training is driven by minimizing a loss function via gradient descent. For multi-class image classification, the **cross-entropy loss** (with a Softmax output) is standard, as it quantifies the discrepancy between the true labels and the predicted probabilities. During training, optimizers like **Stochastic Gradient Descent (SGD)** (often with momentum), **Adam**, or **RMSProp** are used to update weights. Each optimizer has trade-offs: e.g. Adam converges faster initially, while SGD with momentum can generalize better in some cases. It is common to adjust the *learning rate* during training (e.g. using step decay, cosine annealing, or schedulers that reduce the rate on plateaus).

Practitioners also use callbacks: for example, **early stopping** can halt training when validation accuracy ceases to improve, preventing overfitting. Another technique is to **reduce the learning rate on a plateau**, giving the optimizer finer steps once learning slows. These tools – loss functions, optimizers, and training schedules – together enable effective end-to-end learning of the CNN parameters on image data.

11. Data Augmentation and Transfer Learning

- **Data Augmentation:** To enlarge limited image datasets and reduce overfitting, random image transformations are applied during training. Common augmentations include random flips (horizontal/vertical), rotations, crops or translations, and color jittering (random changes in brightness, contrast, saturation). For example, randomly rotating or flipping a food image ensures the CNN sees diverse orientations of the same dish. More advanced techniques include *cutout* (randomly masking a patch) and *mixup* (blending two images), as well as using generative models (GANs) to synthesize entirely new images for rare classes. Overall, augmentation increases dataset diversity and makes the learned features more robust to variations.
- **Transfer Learning:** Given the cost of large datasets, it is common to reuse CNNs pre-trained on large benchmarks (like ImageNet) for new tasks. In **feature extraction**, one keeps the early convolutional layers of a pre-trained model fixed and trains only a new classifier on top. In **fine-tuning**, one unfreezes some top layers of the pretrained model and retrains them (usually with a small learning rate) on the new data. This leverages general visual features (edges, textures) learned from vast data and applies them to the specific domain. For instance, one might take a pretrained VGG16 or ResNet50 and fine-tune it on a subset of the Food-101 dataset. This approach often achieves good performance even with modest data, because the model has already learned useful representations.

Transfer learning and augmentation together form a powerful combination: they mitigate overfitting and improve generalization when training CNNs on limited or specialized image datasets.

12. Model Evaluation

CNN performance on image classification is measured using standard metrics. **Accuracy** (the fraction of correctly classified images) is the simplest metric. However, in many cases precision, recall, and the F1-score (harmonic mean of precision and recall) are also important, especially if classes are imbalanced. A **confusion matrix** is often used to analyze per-class performance (how often each class is confused with others).

For multi-class problems, we may compute *macro*-averaged metrics (treat each class equally) or *micro*-averaged metrics (aggregate over all instances). One can also use one-vs-rest schemes and compute ROC curves and AUC for each class in multi-class settings. In summary, a combination of accuracy, precision/recall/F1, and confusion-matrix analysis provides a comprehensive evaluation of a CNN's classification ability.

13. Real-World Challenges

- **Class Imbalance:** Many real datasets have uneven class distributions (some categories are under-represented). CNNs trained naively on imbalanced data tend to be biased toward majority classes. This requires strategies like oversampling minority classes, using weighted loss functions, or generating synthetic samples for rare classes. Imbalance can lead to high accuracy but poor performance on minority classes, so careful metric selection (e.g. macro-F1) is needed.
- **Overfitting on Small Data:** CNNs have millions of parameters; without enough data they can easily overfit. This is especially problematic when the dataset is small or lacks diversity. Regularization (dropout, L2, augmentation) and transfer learning help, but the risk remains that a CNN "memorizes" the training images and fails to generalize. Monitoring validation performance and using early stopping are important to detect overfitting early.
- **Domain Shift:** Models often see a different distribution of images in deployment than they did in training. For example, a CNN trained on studio-lit food photos may perform poorly on user-taken mobile photos due to changes in lighting, angle, or background. Even differences in image resolution or color can degrade accuracy. Such *domain shift* is a major challenge; techniques like domain adaptation or collecting representative

training data are needed. In one food recognition study, models trained on a controlled dataset had significant performance drops on real-world photos.

- **Inference and Deployment Constraints:** State-of-the-art CNNs can be large and slow. Running them in real-time (e.g. on smartphones or embedded systems) can be difficult. There is a trade-off between model size/latency and accuracy. Deployment often requires model compression (pruning, quantization) or using specialized hardware. Ensuring fast inference and low memory use while maintaining accuracy is an ongoing engineering challenge.

Each of these issues – imbalance, overfitting, domain differences, and computational cost – must be addressed in practical CNN applications. Techniques like augmentation, regularization, transfer learning, and model optimization are used to mitigate these challenges.

14. Advantages and Disadvantages of CNN

(1) Advantages

- CNNs automatically learn features directly from data, eliminating the need for manual feature engineering and often achieving higher accuracy.
- They use far fewer parameters than comparable MLPs due to weight sharing, making training more efficient and reducing overfitting.
- CNNs effectively capture local patterns in images, such as edges and textures, while naturally handling spatial hierarchies and color channels.
- Through convolution and pooling, they are robust to variations in position, scale, and moderate changes in illumination or viewpoint.

(2) Disadvantages

- CNNs require large amounts of labeled data and can overfit severely on small datasets.
- Training deep CNNs is computationally intensive, often requiring GPUs or TPUs, and inference can be slow for large models.
- They are often considered “black boxes,” with limited interpretability of their decision processes.
- With millions of parameters, CNNs are prone to overfitting, especially on small or imbalanced datasets.
- CNNs are vulnerable to adversarial perturbations, where small targeted changes to the input can lead to incorrect predictions.

15. Summary

Convolutional Neural Networks have revolutionized computer vision by enabling end-to-end, hierarchical feature learning from images. CNNs automatically discover spatial features (from edges to objects) through stacked convolutional, pooling, and dense layers, trained with modern optimizers. This has led to unprecedented accuracy in tasks like image classification, detection, and segmentation. As detailed above, successful CNN design combines core components (conv/pooling/activation), regularization techniques, and architecture innovations (e.g. residual connections). The next part of this report will build on this theory: we will implement CNNs for a food recognition task (using a subset of the Food-101 dataset), comparing a custom CNN trained from scratch against a transfer-learning model based on a pretrained architecture. These experiments will highlight how the concepts covered in this theoretical review play out in practice on real image data.

Part 2: Practical Application

1. Problem Description

- **Title**: *Food Image Classification Using Convolutional Neural Networks (CNN) and Transfer Learning*
- **Research objectives:**

The project aims to classify food images based on the *Food-101* dataset by developing and comparing different Convolutional Neural Network (CNN) architectures. The primary objectives are as follows:

- To implement CNN models capable of automatically classifying images into predefined food categories.
- To understand the end-to-end workflow of deep learning model development, including data preprocessing (image resizing, normalization, and train/validation/test splitting), model design, and performance evaluation.
- To build CNN architectures incorporating convolutional, pooling, activation (ReLU), and fully connected layers, and to train them using optimization algorithms such as Stochastic Gradient Descent (SGD) and Adam.
- To evaluate the models' performance using metrics such as Accuracy, Precision, Recall, F1-score, and the Confusion Matrix.

- To explore the use of **Transfer Learning**, leveraging pre-trained CNN models (e.g., VGG16, Xception, ResNet) trained on large-scale datasets like ImageNet, thereby reducing training time and improving generalization.
- To apply **Fine-tuning** techniques to further adapt pre-trained weights to the target dataset and enhance performance.
- To compare the performance between CNNs trained from scratch and CNNs utilizing transfer learning, thereby highlighting differences in efficiency, accuracy, and generalization ability.
- To draw meaningful conclusions on the effectiveness of CNNs and transfer learning for real-world food image classification tasks.

- **Input of the problem:**

- **Image Dataset:** The *Food-101* dataset containing labeled food images, with 20 selected categories used for classification.
- **Pre-trained CNN Models:** Several pre-trained CNN architectures (e.g., VGG16, ResNet50, MobileNet) were initially considered. However, **Xception**, pre-trained on the ImageNet dataset, was selected as the base model for Transfer Learning due to its strong performance and efficient depthwise separable convolutions.
- **Hyperparameters:** Learning rate, batch size, number of epochs, and optimization algorithm (SGD/Adam).

- **Output of the problem:**

Four models were trained and evaluated in this project:

(1) CNN trained from scratch **without data augmentation**
(2) CNN trained from scratch **with data augmentation**
(3) CNN using **Transfer Learning (Xception as base model)** without data augmentation
(4) CNN using **Transfer Learning (Xception as base model)** with data augmentation

The models were evaluated based on:

(1) Quantitative metrics: Accuracy, Precision, Recall, and F1-score on the test set
(2) Training and validation performance: Loss and Accuracy curves
(3) Visual evaluation: Confusion Matrix analysis

The comparison focuses on performance, training time, and generalization capability among the four models, thereby assessing the impact of both **Data Augmentation** and **Transfer Learning** on food image classification.

- **Summary of the tasks performed for this problem.**

In this report, the following tasks were carried out systematically to address the image classification problem using CNN and Transfer Learning:

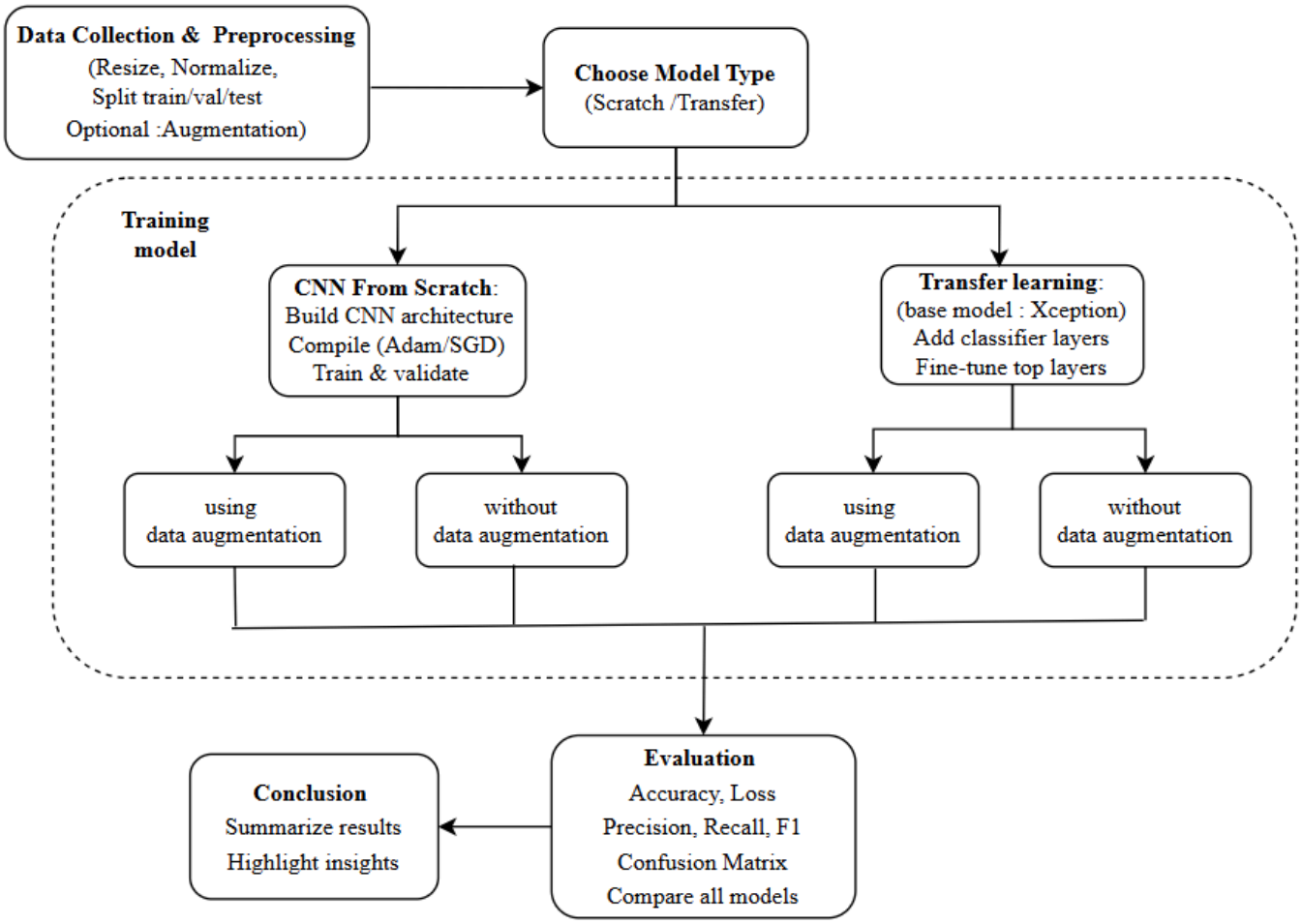

Figure 5: Workflow for CNN Model Training and Evaluation with Transfer Learning

Source: Author's own compilation.

2. Dataset Description

- **Download link** : Food-101 with picking 20 classes

- **Describe the dataset :**

The dataset used in this study is a subset of the **Food-101** collection, which includes images from various food categories. For this project, **20 classes** such as ('guacamole', 'hot_and_sour_soup', 'ravioli', 'caprese_salad', 'chocolate_mousse', 'strawberry_shortcake', 'oysters', 'sashimi', 'poutine', 'deviled_eggs', 'huevos_rancheros', 'prime_rib', 'panna_cotta', 'tacos', 'beef_carpaccio', 'donuts', 'waffles', 'bread_pudding', 'beignets', 'mussels') were selected, with **1,000 images per class**, totaling **20,000 images**. Each image represents different dishes and food types under diverse backgrounds and lighting conditions, ensuring good variability for model training and evaluation. All images were resized to a consistent input dimension (**224×224×3**).

- **Split the dataset into three subsets using an 80:10:10 ratio**

  - **Training set:** 16,000 images (used to train the models)
  - **Validation set:** 2,000 images (used to tune hyperparameters and monitor performance)
  - **Test set:** 2,000 images (used to evaluate final model performance)

3. CNN Model Design

A. Vanilla CNN (No Augmentation, No Transfer Learning)

The model is systematically structured into a feature extraction base and a classification head, processing input image tensors to produce final class probabilities.

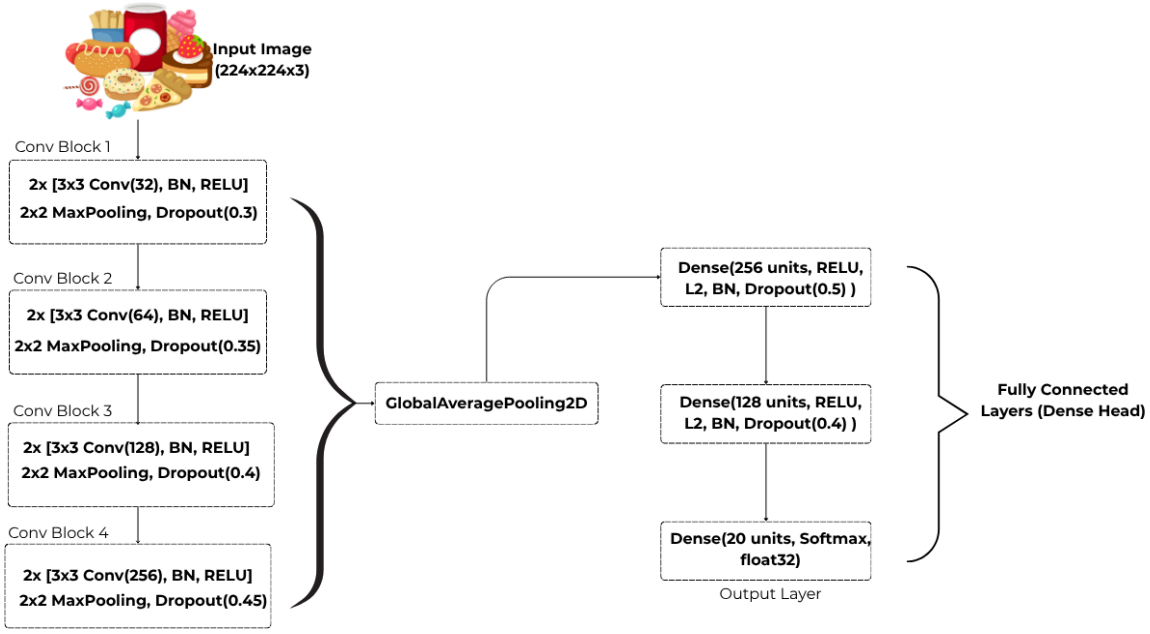

Figure 6: Vanilla CNN Classification Pipeline

Source: Author's own compilation.

1. **Input Layers**

Input Shape: The input tensor to the network is precisely defined with a shape of (224,224,3), corresponding to image dimensions of 224 pixels in height and width, and 3 channels for Red, Green, and Blue (RGB) color representation.

2. **Convolutional Blocks (Feature Extraction Base)**

The core of the feature extractor consists of four sequential blocks. Each block is designed to learn progressively more complex features while downsampling the spatial dimensions.

Block 1:

- Conv2D: 32 filters, (3×3) kernel.
- BatchNormalization: Normalizes activations to stabilize learning.
- Conv2D: 32 filters, (3×3) kernel.
- BatchNormalization: Normalizes activations.
- MaxPooling2D: A (2×2) pooling operation is applied, downsampling the feature map.
- Dropout: A dropout rate of 0.3 is applied to randomly deactivate 30% of neurons, preventing co-adaptation of feature detectors.

Block 2:

- Conv2D: 64 filters, (3×3) kernel.
- BatchNormalization: Normalizes activations.
- Conv2D: 64 filters, (3×3) kernel.
- BatchNormalization: Normalizes activations.
- MaxPooling2D: (2×2) pooling.
- Dropout: The dropout rate is incrementally increased to 0.35 to enhance regularization.

Block 3:

- Conv2D: 128 filters, (3×3) kernel.
- BatchNormalization: Normalizes activations.
- Conv2D: 128 filters, (3×3) kernel.
- BatchNormalization: Normalizes activations.
- MaxPooling2D: (2×2) pooling.
- Dropout: The dropout rate is further escalated to 0.4 for stronger regularization.

Block 4:

- Conv2D: 256 filters, (3×3) kernel.
- BatchNormalization: Normalizes activations.
- Conv2D: 256 filters, (3×3) kernel.
- BatchNormalization: Normalizes activations.
- MaxPooling2D: (2×2) pooling.
- Dropout: The highest dropout rate within the feature extractor, 0.45, is applied to this block.

3. Fully Connected Layers (Dense Head)

After feature extraction, the model transitions to a dense classifier head to make the final prediction.

- *GlobalAveragePooling2D*: This layer serves as an efficient bridge between the convolutional and dense layers. It replaces a traditional Flatten layer by averaging the spatial dimensions of the feature maps, which significantly reduces the number of parameters and makes the model more robust to spatial translations of features.
- *Dense Layers*:
    - A Dense layer with 256 units, 'relu' activation, and L2 regularization. It is followed by BatchNormalization and Dropout (0.5).

    - A second Dense layer with 128 units, 'relu' activation, and L2 regularization. It is also followed by BatchNormalization and Dropout (0.4).
- *Output Layer*: The final Dense layer has a number of units equal to the number of classes (20). It employs a 'softmax' activation function to output a probability distribution over the classes. The dtype is explicitly set to 'float32' to ensure stable probability outputs when using mixed precision.

4. Regularization and Optimization Strategies

The model's training is governed by a robust suite of strategies designed to ensure stable convergence, prevent overfitting, and maximize computational efficiency.

- L2 Regularization: A kernel L2 regularizer with a penalty of 10^{-3} is applied to all convolutional and dense layers. L2 regularization discourages model complexity by penalizing large weights, which improves generalization.
- Batch Normalization: Used after every convolutional and dense layer (except the output), BatchNormalization normalizes activations to stabilize and accelerate training by mitigating internal covariate shift.
- Dropout: Dropout is strategically applied after each MaxPooling2D layer and between the dense layers in the classification head, with progressively increasing rates from 0.3 to 0.5.
- Performance and Optimization
    - Mixed Precision Training: The mixed_float16 policy is used to leverage modern GPU Tensor Cores. This accelerates computation and reduces memory consumption, enabling more efficient training.
    - AdamW Optimizer: The model is optimized using AdamW, an advanced version of the Adam optimizer that decouples weight decay from the adaptive learning rate mechanism, which often leads to better model generalization.
    - Label Smoothing: The CategoricalCrossentropy loss function is augmented with a label smoothing factor of 0.15. This encourages the model to be less confident in its predictions by softening the one-hot encoded target labels, which improves model calibration and robustness.
- Callbacks for Training Control
    - EarlyStopping: This callback monitors the validation loss. If no improvement is seen for 25 consecutive epochs, training is automatically halted, and the model weights from the best-performing

epoch are restored. This prevents overfitting and saves computational resources.

- ReduceLROnPlateau: This callback also monitors the validation loss. If the loss plateaus for 8 epochs, the learning rate is reduced by a factor of 0.5. This allows the model to make finer adjustments and descend more carefully into a minimum in the loss landscape.
- LearningRateScheduler (Cosine Decay): This callback implements a smooth cosine annealing schedule for the learning rate. It gradually decays the learning rate from an initial value of 10^{-3} to a minimum of 10^{-7} over the course of 100 epochs, providing a more effective learning rate progression than a simple step decay.

B. CNN with Data Augmentation Only

1. Input Layer and Data Augmentation

Input Specification: The model is configured to accept input images with dimensions of (160, 160, 3), representing the height, width, and RGB color channels, respectively.

Data Augmentation (Training Phase): A key component of the regularization strategy involves applying a rich set of on-the-fly transformations to the training images. This process artificially expands the dataset's diversity, compelling the model to learn more invariant features. The applied augmentations include:

- *Normalization*: Rescaling pixel values from the [0, 255] range to [0, 1].
- *Geometric Transformations*: Random rotations up to 30 degrees, horizontal and vertical shifts of up to 25% of the image dimension, shearing transformations (shear range of 0.2), and zooming up to 30%.
- *Photometric Transformations*: Random adjustments to brightness (within a range of [0.6, 1.4]) and color channels (up to a shift of 30.0).
- *Flipping*: Random horizontal flips.
- *Pixel Filling*: Newly created pixels resulting from transformations are filled using reflection of the boundary pixels (fill_mode='reflect').

Validation and test data are subjected only to pixel value normalization to ensure consistent evaluation.

2. Convolutional Blocks

The core of the network is the feature extraction base, which consists of four sequential blocks. Each block employs three consecutive Conv2D layers, each followed by BatchNormalization. This design deepens the network at each stage,

allowing for the learning of increasingly complex feature hierarchies before spatial downsampling occurs via MaxPooling2D.

Block 1 (64 Filters)

- 3x Conv2D: Three convolutional layers, each applying 64 filters of size (3×3) with 'same' padding and 'relu' activation. The initial layer also defines the model's input_shape.
- 3x BatchNormalization: Normalizes activations following each convolutional layer to stabilize and accelerate training.
- MaxPooling2D (2x2): Reduces spatial dimensions by a factor of two.
- Dropout (0.25): Applies a 25% dropout rate for regularization.

Block 2 (128 Filters)

- 3x Conv2D: Three convolutional layers, each with 128 filters of size (3×3), 'same' padding, and 'relu' activation.
- 3x BatchNormalization: Applied after each Conv2D layer.
- MaxPooling2D (2x2): Further spatial downsampling.
- Dropout (0.3): The dropout rate is increased to provide stronger regularization for this deeper block.

Block 3 (256 Filters)

- 3x Conv2D: Three convolutional layers, each with 256 filters of size (3×3), 'same' padding, and 'relu' activation.
- 3x BatchNormalization: Applied after each Conv2D layer.
- MaxPooling2D (2x2): Spatial downsampling.
- Dropout (0.35): The dropout rate is again increased in proportion to the block's increased capacity.

Block 4 (512 Filters)

- 3x Conv2D: Three convolutional layers, each with 512 filters of size (3×3), 'same' padding, and 'relu' activation.
- 3x BatchNormalization: Applied after each Conv2D layer.
- MaxPooling2D (2x2): Final spatial downsampling operation.
- Dropout (0.4): The highest dropout rate within the feature extractor, applied to the block with the most parameters.

3. Fully Connected Layers (Dense Head)

After hierarchical feature extraction, the resulting feature maps are passed to a dense head for final classification.

- *GlobalAveragePooling2D*: This layer serves as an efficient bridge between the convolutional and dense parts of the network. By averaging the spatial

dimensions of the final feature maps, it produces a fixed-size feature vector, drastically reduces the number of trainable parameters, and enhances the model's robustness to spatial variations in the input.

- *Dense (512 units*): A fully connected layer with 512 neurons and 'relu' activation, responsible for learning non-linear combinations of the high-level features.
- *Dropout (0.5)*: A substantial 50% dropout rate is applied to this layer to provide strong regularization for the classifier.
- *Dense (NUM_CLASSES units)*: The final output layer contains a neuron for each of the 20 food classes. It employs a 'softmax' activation function to generate a probability distribution over the classes. The dtype is explicitly set to 'float32' to ensure stable probability outputs, a crucial practice when utilizing mixed precision training.

4. Regularization and Optimization Strategies

The model's training regimen is governed by a carefully selected set of modern optimization and regularization techniques to ensure efficient, stable, and effective learning.

- *BatchNormalization*: Utilized after every convolutional layer to normalize layer activations, which stabilizes the training process and accelerates convergence.
- *Dropout*: Strategically deployed after each pooling operation and in the dense head with progressively increasing rates (0.25 to 0.5). This technique is essential for preventing the co-adaptation of neurons and improving model generalization.
- *Mixed Precision Training*: Employs the mixed_float16 policy to leverage GPU Tensor Cores, resulting in faster training epochs and reduced memory consumption.
- *AdamW Optimizer*: An advanced version of the Adam optimizer that decouples weight decay (set to 1e-4) from the adaptive learning rate updates, often leading to improved generalization. The initial learning rate is set to 3e-4.
- *Label Smoothing (0.1)*: Applied to the CategoricalCrossentropy loss function. This technique discourages the model from making overconfident predictions, thereby improving its calibration and robustness.
- *Callbacks*:

- *EarlyStopping*: Monitors the validation accuracy (val_accuracy) and terminates training if no improvement occurs for 15 consecutive epochs, restoring the model weights from the best-performing epoch.
- *ReduceLROnPlateau*: Monitors the validation loss (val_loss) and reduces the learning rate by a factor of 0.5 if it stagnates for 5 epochs, with a minimum learning rate of 1e-6.
- *LearningRateScheduler (One-Cycle Policy)*: Implements a one-cycle learning rate schedule. The learning rate increases linearly from a low value to a maximum (max_lr of 3e-3) over the first 30% of epochs and then anneals linearly over the remaining 70%, a strategy known to promote faster convergence and superior performance.

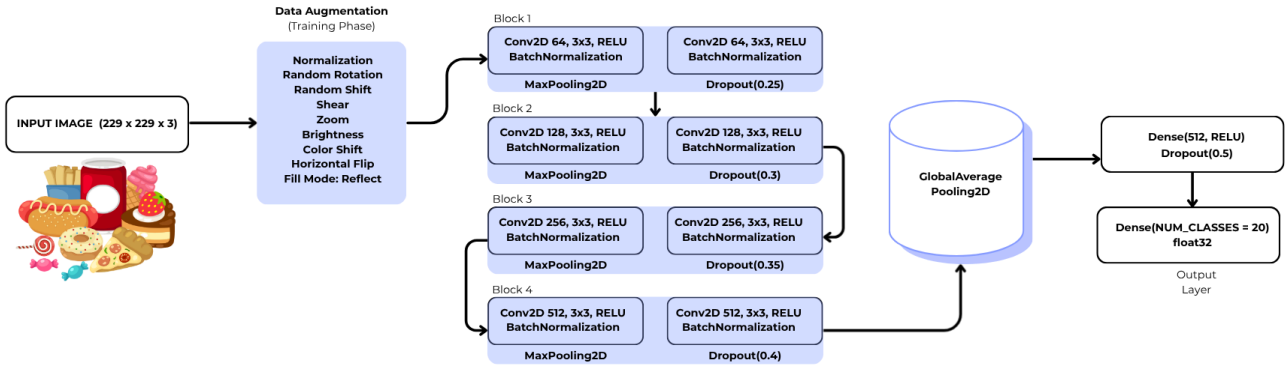

Figure 7: Detailed Architecture of the VGG-mini Based CNN

Source: Author's own compilation.

C. Transfer-Learning CNN (No Augmentation)

1. Transfer Learning Model: Xception

This approach leverages the pre-trained Xception network as a powerful feature extractor, with a custom classification head appended for the specific task. The model is trained using a two-stage fine-tuning strategy.

Xception (Extreme Inception) is a state-of-the-art CNN architecture that modifies Inception modules by using depthwise separable convolutions. This design is highly effective for several reasons:

Efficiency: It separates the process of learning spatial correlations (per channel) and cross-channel correlations (via 1x1 convolutions). This drastically reduces computational cost and the number of parameters compared to standard convolutions, leading to a more efficient model.

Performance: Despite its efficiency, Xception achieves top-tier accuracy on large-scale benchmarks like ImageNet.
Transferability: By loading weights pre-trained on the diverse ImageNet dataset, the model inherits a rich set of general-purpose visual features (e.g., edges, textures, shapes). This knowledge is highly transferable to related domains like food classification, enabling the model to achieve high performance with less data and training time.

a. Architectural Specification

Input and Preprocessing: The Xception base requires an input shape of (299, 299, 3). Images are preprocessed using the tf.keras.applications.xception.preprocess_input function, which scales pixel values to the range [-1, 1].
Base Model: The Xception network is loaded with weights='imagenet' and with its top classification layer removed (include_top=False).
Custom Classification Head: A new dense head is stacked on top of the Xception base:

- GlobalAveragePooling2D: Flattens the feature maps from the base model.
- Dense (512 units): A fully connected layer with ReLU activation, followed by BatchNormalization and Dropout (0.5).
- Dense (256 units): A second fully connected layer with ReLU activation, followed by BatchNormalization and Dropout (0.4).
- Output Layer: A final Dense layer with 20 neurons and 'softmax' activation.

b. Two-Stage Training Strategy

The model is trained in two distinct phases to ensure stable learning.
Stage 1: Feature Extraction (6 Epochs):
The layers of the Xception base_model are frozen (trainable = False). Only the weights of the newly added custom head are trained. This allows the new classifier to learn how to interpret the powerful, pre-existing features from Xception without disrupting them. The model is compiled with an Adam optimizer (learning rate 10−4) and CategoricalCrossentropy loss with label smoothing (0.05).
Stage 2: Fine-Tuning (30 Epochs):
The top ~100 layers of the Xception base_model are unfrozen (trainable = True). This allows the model to make small, specific adjustments to the high-level pre-trained features to better adapt them to the nuances of the food dataset. The model is recompiled with a much lower learning rate (Adam, lr=10 −5) to prevent catastrophic forgetting of the valuable pre-trained weights.

c. Optimization and Checkpointing

Throughout both training stages, EarlyStopping (patience=8) and ReduceLROnPlateau (patience=3) are used to manage the training process. A ModelCheckpoint callback is also employed to save the best model weights based on val_loss at the end of each stage.

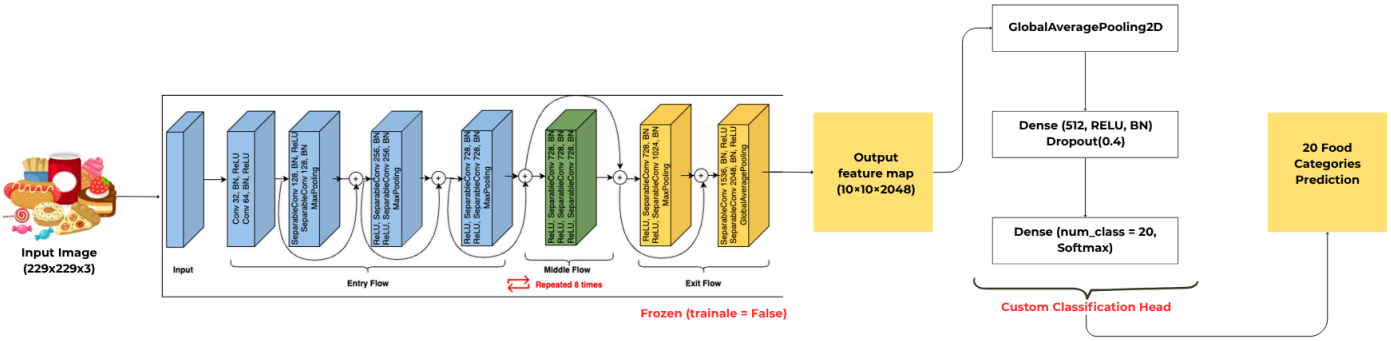

Figure 8: Custom CNN Classification Pipeline

Source: Author's own compilation.

D. Transfer-Learning CNN with Data Augmentation

The model employs **Transfer Learning** with the **Xception** backbone pretrained on **ImageNet**, using its convolutional base as a feature extractor while replacing the top layers with a custom classification head for 20 food categories. To enhance generalization, **Data Augmentation** (rotation, flip, scaling, brightness) is applied during training. This combination leverages pretrained knowledge and synthetic data diversity to boost performance and reduce overfitting on the *food20_split* dataset.

1. Base Architecture — Pretrained Xception

The **base model** corresponds to the full Xception network (Extreme Inception) excluding its final classification head. Xception, designed by **François Chollet** (the creator of Keras), is considered an advanced evolution of the Inception family and is often described as an *"Inception fully based on depthwise separable convolutions."*

This conceptually means that instead of performing standard convolutions (which simultaneously learn spatial and cross-channel correlations), Xception **decouples** these processes:

- **Depthwise Convolution** performs spatial filtering independently on each input channel.
- **Pointwise Convolution (1×1 convolution)** then combines these filtered outputs across channels.

This separation significantly **reduces the number of parameters**, **improves efficiency**, and **enhances representational power**, making the model deeper yet computationally manageable.

Structurally, the Xception architecture consists of **three major flows**:

- **Entry Flow:**
  Begins with standard convolutions to process the input image, followed by a series of *SeparableConv2D* → *BatchNormalization* → *ReLU* blocks interleaved with *MaxPooling*.
  Each step progressively reduces the spatial dimensions (height and width) while increasing the feature depth.
- **Middle Flow:**
  Composed of **eight identical residual blocks**, each containing three *SeparableConv2D* layers connected via skip connections.
  This section captures deep non-linear representations while maintaining training stability through residual links.
- **Exit Flow:**
  This final stage refines the features and expands the channel depth up to **2048 filters**, preparing them for global pooling.

For an input of **299×299×3**, the final output feature map from the base network typically has a shape around **(10×10×2048)**.

2. Custom Classification Head (Top Layers)

After the feature extraction stage, a **custom classifier head** is added to map these high-level representations into probabilities corresponding to the 20 food categories.

The custom head includes the following sequence:

- **GlobalAveragePooling2D:**
  Transforms the 10×10×2048 feature map into a 2048-dimensional vector by

averaging each channel's spatial information.
This operation drastically reduces parameters and ensures that the model retains global spatial context

- **Dense(512, activation='relu'):**
  A fully connected layer with 512 neurons.
  It learns complex, non-linear combinations of the extracted features.
- **BatchNormalization:**
  Stabilizes the activations by normalizing feature distributions, which helps in faster and more reliable convergence.
- **Dropout(0.5):**
  Randomly drops 50% of neurons during training to reduce overfitting and improve generalization.
- **Dense(256, activation='relu'):**
  A second fully connected layer to refine the learned representations and gradually compress feature space.
- **BatchNormalization + Dropout(0.4):**
  A second normalization and regularization phase that further stabilizes training.
- **Dense(num_classes=20, activation='softmax'):**
  The final classification layer producing class probabilities across 20 output categories.

This custom head introduces approximately **1.18 million trainable parameters**, while the frozen Xception base contains around **20.86 million non-trainable parameters**, totaling roughly **22 million parameters** in the whole model.

3. Training Configuration

The training strategy is divided into **two major stages**, common in transfer learning workflows:

Stage 1 — Feature Extraction

- The Xception base is **frozen** (trainable=False), ensuring that only the new classifier head is trained.
- Optimizer: **Adam (learning rate = 1e-4)**

- Loss Function: **Categorical Crossentropy** with **label smoothing = 0.05**, which helps prevent overconfidence in predictions and mitigates noise from potentially mislabeled data.
- Callbacks include **EarlyStopping**, **ReduceLROnPlateau**, and **ModelCheckpoint** to monitor and optimize training efficiency.

Stage 2 — Fine-tuning (Optional)

Typically involves unfreezing some deeper layers of the Xception base and training with a smaller learning rate (**1e-5**) to allow the pretrained filters to adjust slightly to the new dataset. For real fine-tuning, selective unfreezing of the final few convolutional blocks is recommended.

4. Model Summary and Parameter Breakdown

From the model.summary() output:

Category	**Parameters**
Total parameters	~22,050,108
Trainable parameters	~1,187,092
Non-trainable parameters	~20,863,016

Table 2 : Model Parameter Distribution

Source: Author's own compilation.

This table provides a clear separation between the *frozen* pretrained layers and the *trainable* custom classification head:

- **Total parameters (~22.0M):**
  Represent the complete model, including both the pretrained Xception backbone and the new classification head. This total parameter count gives an idea of the overall model complexity and memory requirements during inference.
- **Non-trainable parameters (~20.86M):**
  These belong entirely to the Xception base network pretrained on ImageNet.

During training, these layers are **frozen** (i.e., their weights are not updated), preserving the powerful visual feature representations already learned. This ensures stable convergence and prevents catastrophic forgetting when adapting to the new dataset.

- **Trainable parameters (~1.19M):**
  Correspond to the **custom fully connected head** added on top of the frozen base. These layers are responsible for learning dataset-specific patterns — in this case, visual distinctions among the 20 food categories. Despite representing less than 6% of the total parameters, this trainable portion determines the model's final classification performance on the target domain.

This configuration effectively balances transfer efficiency and training flexibility. By freezing most of the Xception base and training only the new dense layers, the model achieves faster convergence and greater stability. The pretrained backbone retains its strong general features from ImageNet, reducing overfitting risks on smaller datasets. As a result, the network functions as a powerful feature extractor and lightweight classifier, well-suited for domain adaptation with limited data and computational resources.

5. Design Rationale

Several design choices demonstrate deliberate engineering for efficiency and generalization:

- **Using Xception as base:** provides highly expressive image representations due to its depthwise separable convolution mechanism.
- **Global Average Pooling:** replaces Flatten to minimize overfitting and ensure a compact, generalizable representation.
- **Two Fully Connected Layers (512 → 256):** balance depth and complexity, adding enough non-linearity without over-parameterization.
- **Dropout + BatchNormalization:** work jointly to stabilize learning and avoid overfitting.
- **Label Smoothing:** softens target distributions, improving robustness against label noise and helping the model avoid excessive confidence.

In essence, this model demonstrates how transfer learning with a robust architecture like Xception bridges the gap between large-scale generic feature learning and

specialized, domain-specific image classification. It effectively integrates computational efficiency, training stability, and strong representational power — qualities that make it a cornerstone in state-of-the-art deep learning applications.

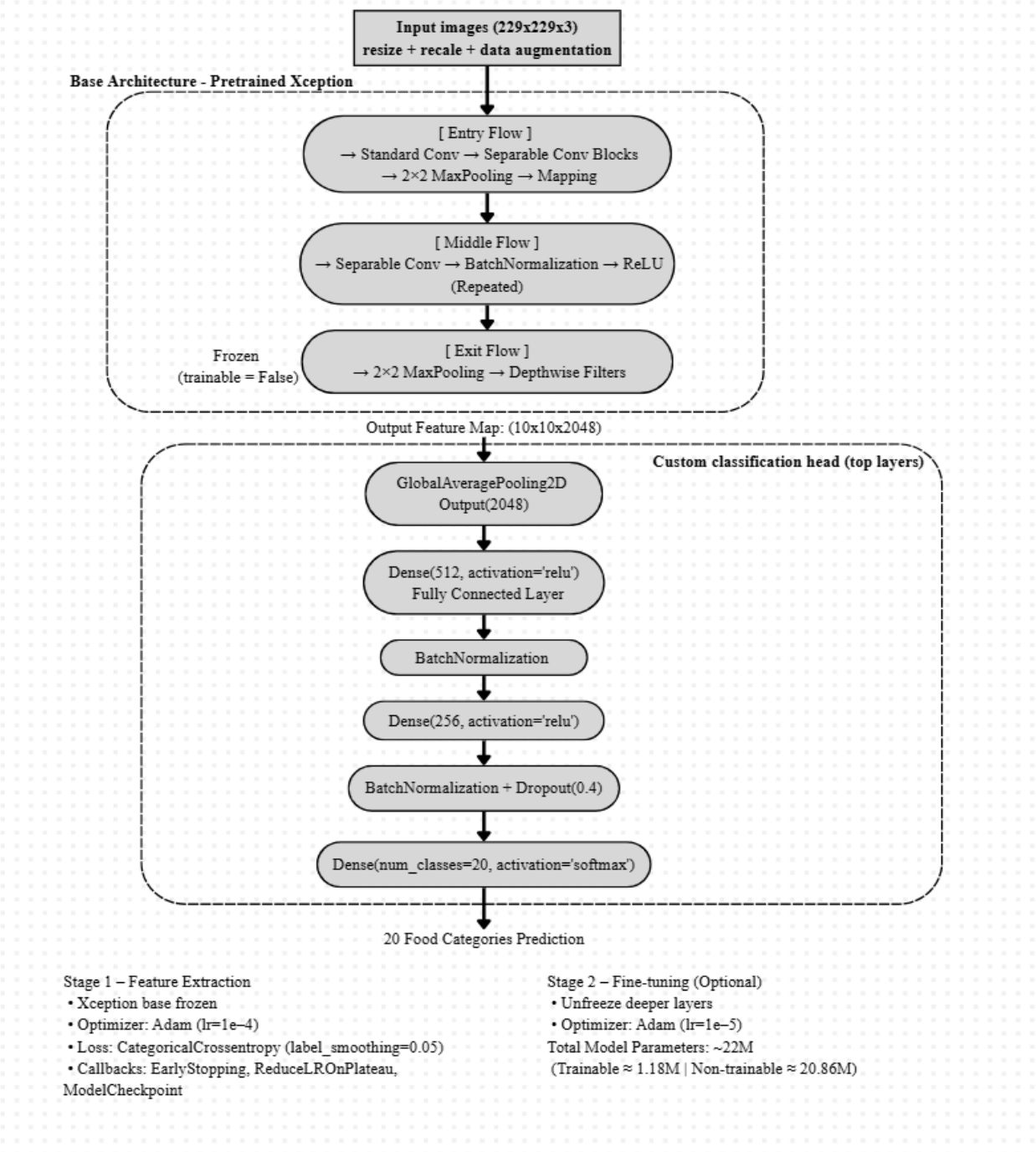

Figure 9: Xception-based Transfer Learning Model with data augmentation

Source: Author's own compilation.

4. Experimental Results

We evaluated the performance of four models on the test set using standard classification metrics: **Accuracy**, **Precision**, **Recall**, and **F1-score**. The models include a baseline CNN, CNN with data augmentation, a CNN with **transfer learning**, and transfer learning with data augmentation. The results are summarized in Table 1.

Model	Accuracy	Precision	Recall	F1-score
CNN	0.8135	0.8163	0.8135	0.8123
CNN + Aug	0.845	0.8525	0.845	0.8458
Trans	0.878	0.8806	0.878	0.8784
Trans + Aug	0.9025	0.9043	0.9025	0.9029

Table 3 : Performance comparison of models on the test set

Source: Author's own compilation.

The experimental results show that transfer learning significantly outperforms the baseline CNN across all metrics, demonstrating the advantage of using pretrained models to extract robust and generalizable features. Data augmentation further improves performance by increasing training data diversity and reducing overfitting, with the combination of transfer learning and augmentation achieving the highest accuracy (90.25%), precision (90.43%), recall (90.25%), and F1-score (90.29%). These findings indicate that for food image classification, leveraging pretrained models along with data augmentation provides an efficient and effective approach to handle limited datasets while achieving state-of-the-art performance.

5. Conclusion

In this study, we investigated and compared the performance of four food image classification architectures on a 20-class subset of Food-101: (1) Vanilla CNN without augmentation, (2) CNN with data augmentation, (3) Transfer Learning using Xception without augmentation, and (4) Transfer Learning with Xception combined with data augmentation. The dataset consisted of 20,000 images (1,000 images per class), split into training, validation, and test sets in an 80:10:10 ratio. Images were preprocessed to match each model's input size (224×224×3 or 299×299×3). Training strategies included regularization techniques (L2, BatchNorm, Dropout), mixed

precision, Adam/AdamW optimizers, label smoothing, and callbacks (EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, and learning-rate schedulers) to ensure stable convergence and reduce overfitting.

Experimental results on the test set show that the Transfer Learning model with augmentation achieved the best performance (Accuracy = 0.9025, Precision = 0.9043, Recall = 0.9025, F1-score = 0.9029). Transfer Learning alone also outperformed the custom CNN models (Trans: Accuracy = 0.878 compared to CNN + Augmentation = 0.845 and baseline CNN = 0.8135). Data augmentation significantly improved performance for both CNN and transfer learning models, indicating that augmentations help the model learn robust features invariant to variations in image angle, lighting, and background.

From these results, we draw three key conclusions: (1) Using a pretrained network (Xception) provides highly transferable and generalizable visual features, improving accuracy when labeled data are limited; (2) Data augmentation is an effective strategy for reducing overfitting and enhancing model generalization; (3) Combining Transfer Learning with augmentation yields synergistic benefits, achieving the highest performance with reasonable training costs compared to training a deep network from scratch.

The study also identifies several limitations and future directions. Limitations include: (i) using only a 20-class subset of Food-101, which limits food diversity and may reduce generalization to out-of-distribution data; (ii) not exploring advanced techniques such as ensembling, deeper layer-wise fine-tuning, or modern vision transformer architectures; (iii) lacking experiments on inference speed and computational cost for real-world deployment. Future work could address these limitations by expanding the dataset (adding more classes and locally collected images), exploring ensembling or pruning methods to balance performance and latency, applying explainability techniques (e.g., Grad-CAM, LIME) to analyze model errors, and evaluating deployment on mobile or edge devices.

In summary, this study demonstrates that combining Transfer Learning (Xception) with data augmentation is an effective and practical solution for improving food image classification accuracy on moderately sized datasets. The results are not only academically valuable but also promising for practical

applications such as food recognition, nutrition recommendation systems, and culinary image analysis.

References

(1) Zhao, Z., Wang, P., & Lu, W. (2021). Multi-layer fusion neural network for deepfake detection. *International Journal of Digital Crime and Forensics, 13*(4), 26–39. https://doi.org/10.4018/IJDCF.20210701.oa3

(2) Keita, Z. (2023, November 14). *An introduction to convolutional neural networks (CNNs)*. DataCamp. https://www.datacamp.com/tutorial/convolutional-neural-networks

(3) Lin, H., Shi, Z., & Zou, Z. (2017). *Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network*. Remote Sensing, 9(5), 480. https://doi.org/10.3390/rs9050480

(4) Tripathi, P. (2021). Transfer learning on deep neural network: A case study on Food-101 food classifier. *International Journal of Engineering Applied Sciences and Technology*, 5(9), 229–232. https://doi.org/10.33564/IJEAST.2021.V05I09.037

(5) Bu, L., Hu, C., & Zhang, X. (2024). Recognition of food images based on transfer learning and ensemble learning. *PLOS ONE*, 19(1), e0296789. https://doi.org/10.1371/journal.pone.0296789

(6) Gomes, D. (2023). Classification of food objects using deep convolutional neural network using transfer learning. *SSRN*. https://doi.org/10.2139/ssrn.5184398