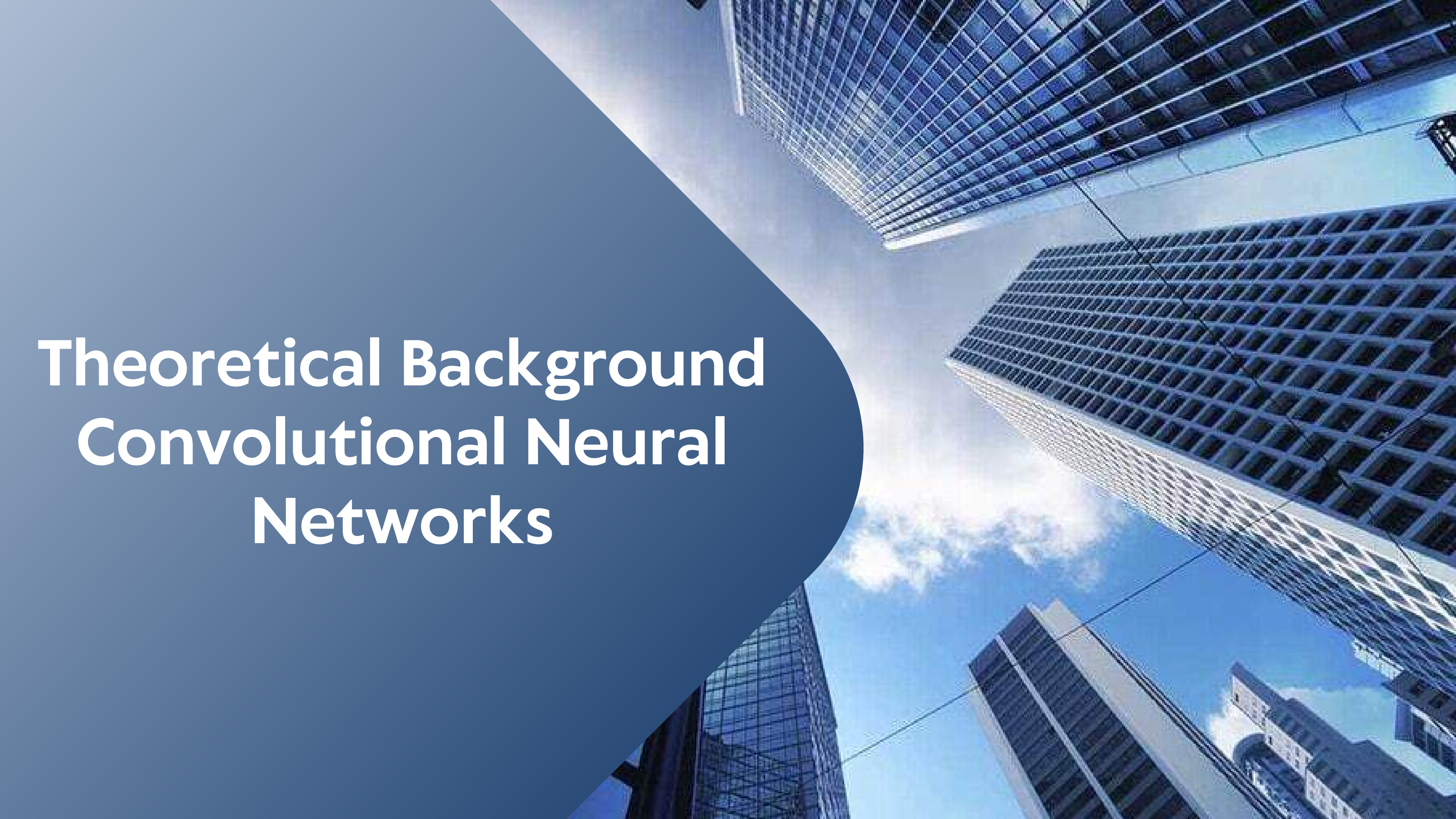


The background features several abstract, rounded geometric shapes in various shades of blue. A large, dark blue shape is in the top left corner. A medium blue shape is in the bottom left corner. A light blue shape is in the bottom right corner. A very light blue shape is in the middle left area. The text is centered on a white background.

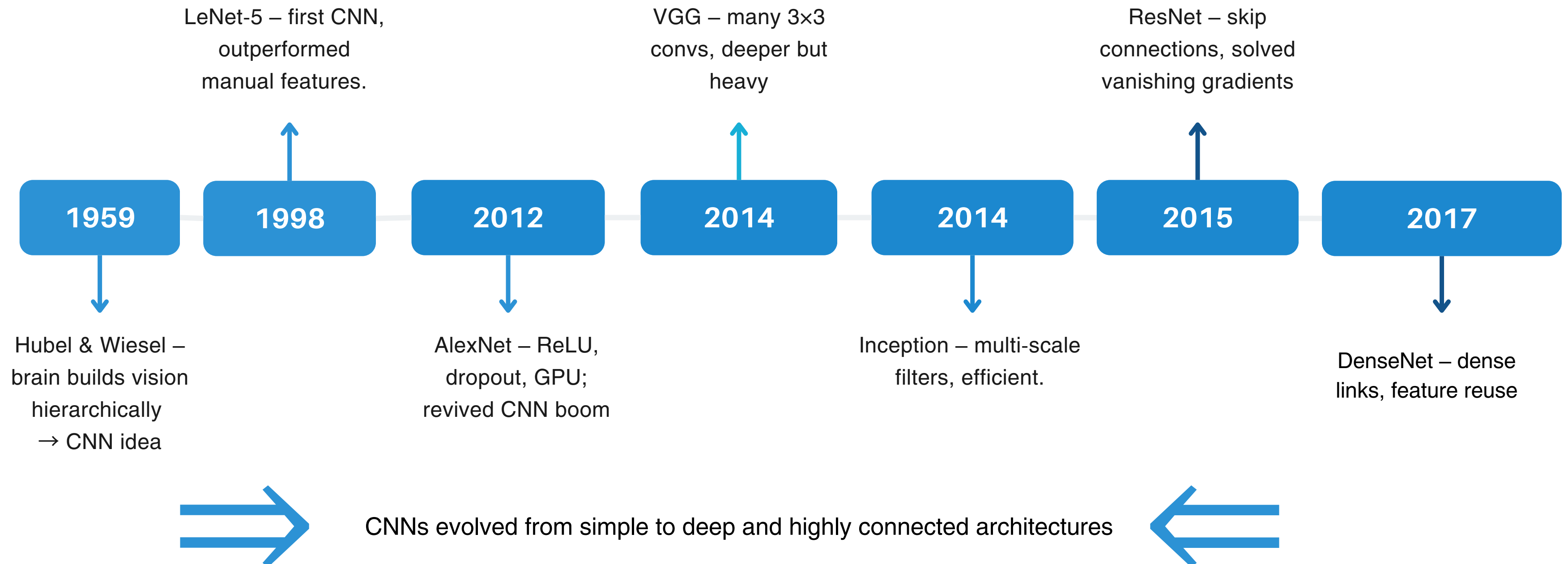
# **FOOD IMAGES CLASSIFICATION**

Group 4

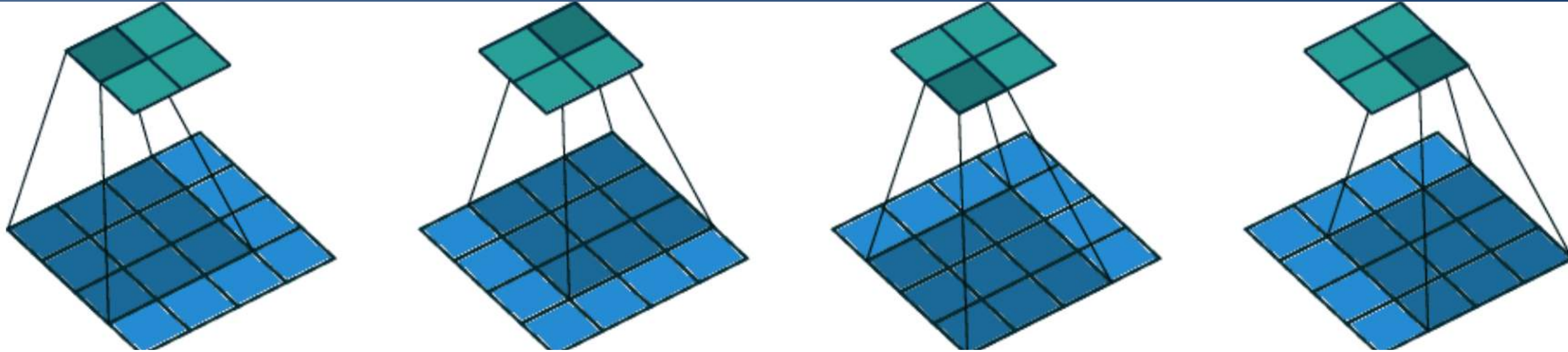
A low-angle, upward-looking photograph of several modern skyscrapers with glass facades, reaching towards a bright blue sky with scattered white clouds. The perspective creates a sense of height and scale. A large, dark blue, semi-circular graphic element is overlaid on the left side of the image, containing the title text in white.

# Theoretical Background Convolutional Neural Networks

# History of Convolutional Neural Networks



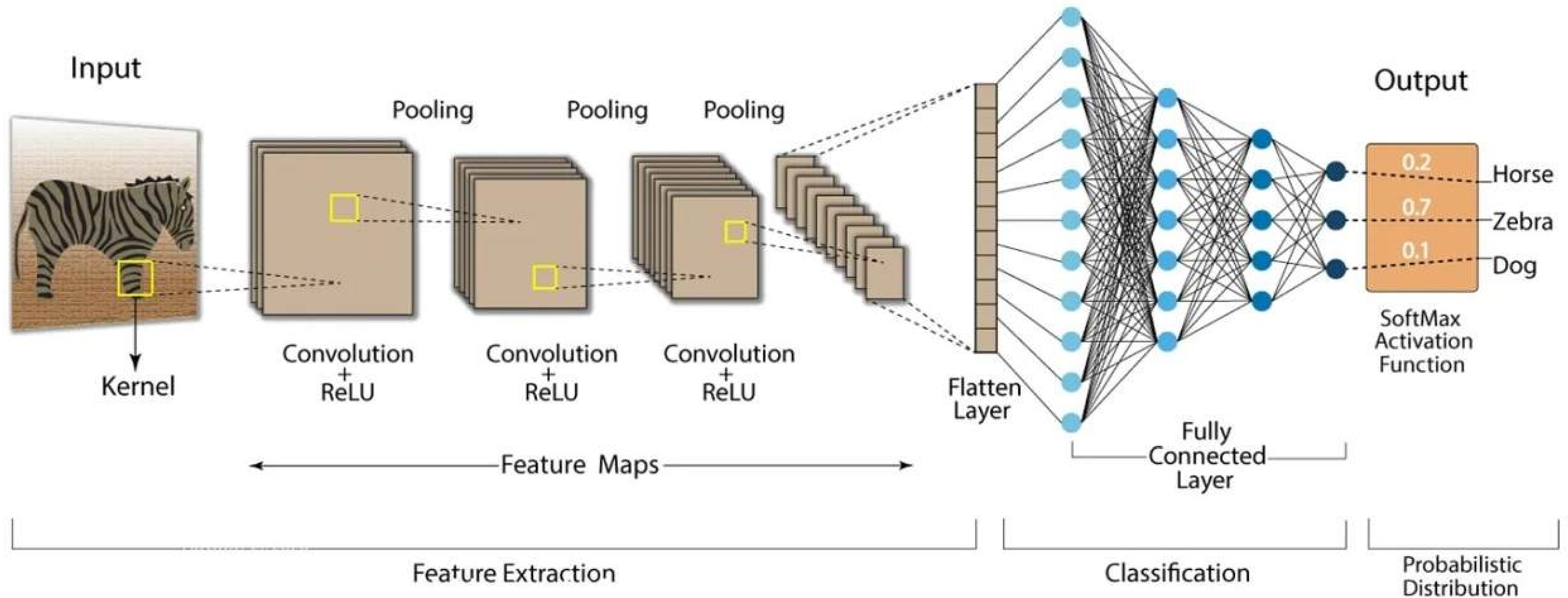
# Introduction of CNN



## Definition of CNN

A Convolutional Neural Network (CNN) is a deep learning architecture specialized for processing image-like data by using convolutional layers to capture spatial and local features.

# Architecture of CNN



# Key features of CNN

- Learns visual patterns automatically from raw pixels
- Builds hierarchical feature representations
- Maintains spatial structure through convolutions
- Generalizes effectively across image variations
- Excels in tasks like classification, detection, and segmentation



## Loss Function

Cross-Entropy + Softmax (multi-class classification)

## Optimizers

SGD, Adam, RMSProp (faster convergence)

## Data Augmentation

Rotation, flipping, cropping, brightness adjustment  
(reduce overfitting)

## Transfer Learning

Use pretrained models (e.g., Xception, VGG16)  
for faster and more efficient learning

# Improvements in CNN training

# Why CNNs Are Suitable for Image Processing

- ◆ Learn Local Features Automatically
- ◆ Translation & Rotation Invariance
- ◆ Preserve Spatial Structure
- ◆ Efficient & Scalable

# Comparison with Traditional MLP

<i>Aspect</i>	Traditional MLP	CNN
Input	Flattens image → loses spatial info	Preserves 2D spatial structure
Feature	Manual or shallow learned features	Automatically extracts spatial features
Parameters	Many weights	Fewer via weight sharing
Performance	Struggles with complex images	Scales well, performs better
Use Case	Generic model	Specialized for images

# PRACTICAL APPLICATION

## Problem

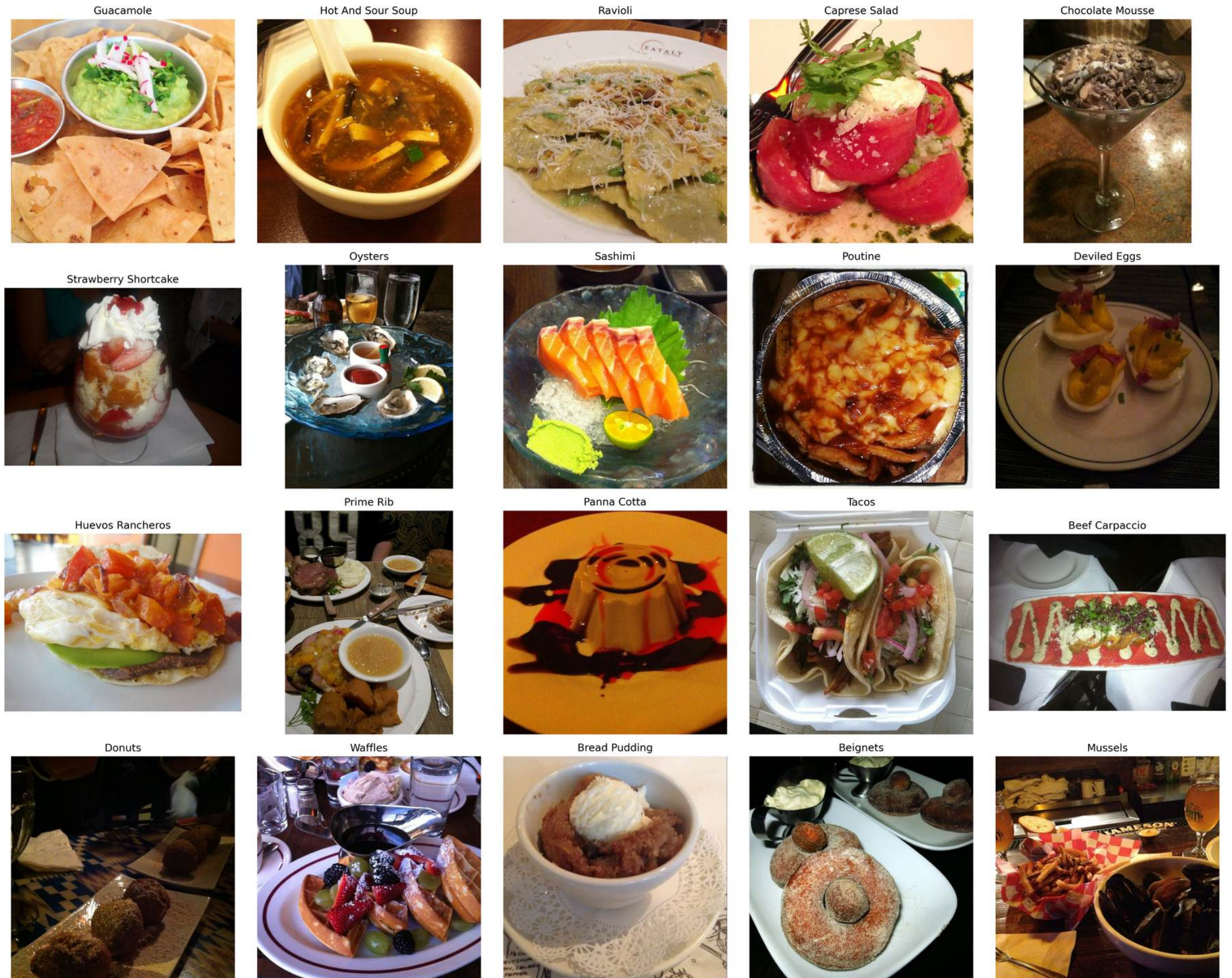
Classify food images (using a Food-101 subset)

## Conclusion

Summary of key results  
achieved

# Dataset

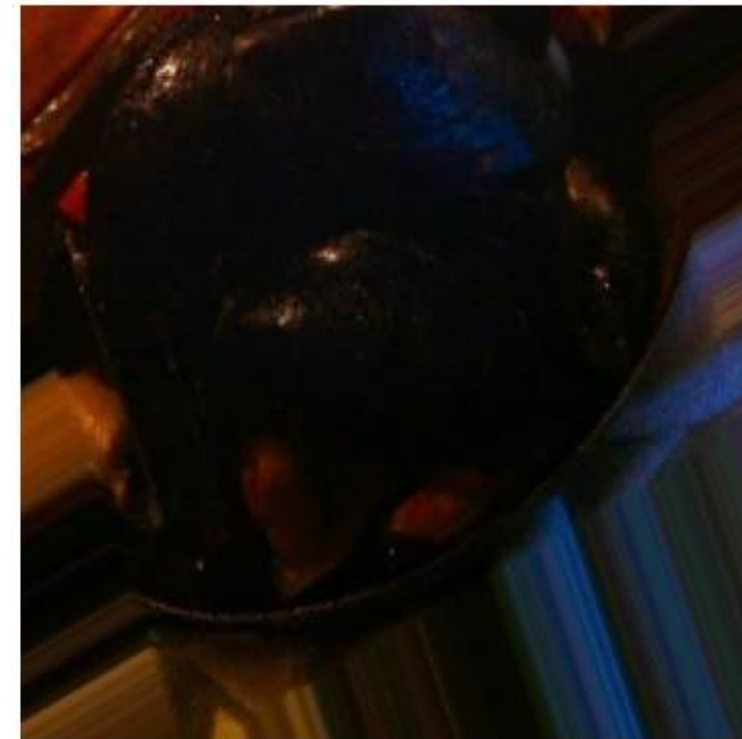
- Classify food images (using a Food-101 subset)
  - 20 classes, 1,000 images/class  
→ 20,000 images total.
  - Dimensions: 224×224×3
- Data Split:  
80% Train – 10% Validation – 10% Test



# Preprocessing

- **Resize:** Standardize (224×224)
- **Normalize:** Scale from [0, 255] to the [0, 1] range.
- **Data Augmentation:**
  - This forces the model to learn more robust features and helps prevent overfitting.
- Random rotation, shifting, zooming, shearing, flipping, and brightness adjustment.

Data Augmentation Examples



# Basic CNN (Vanilla CNN) Design

Model Architecture Diagram

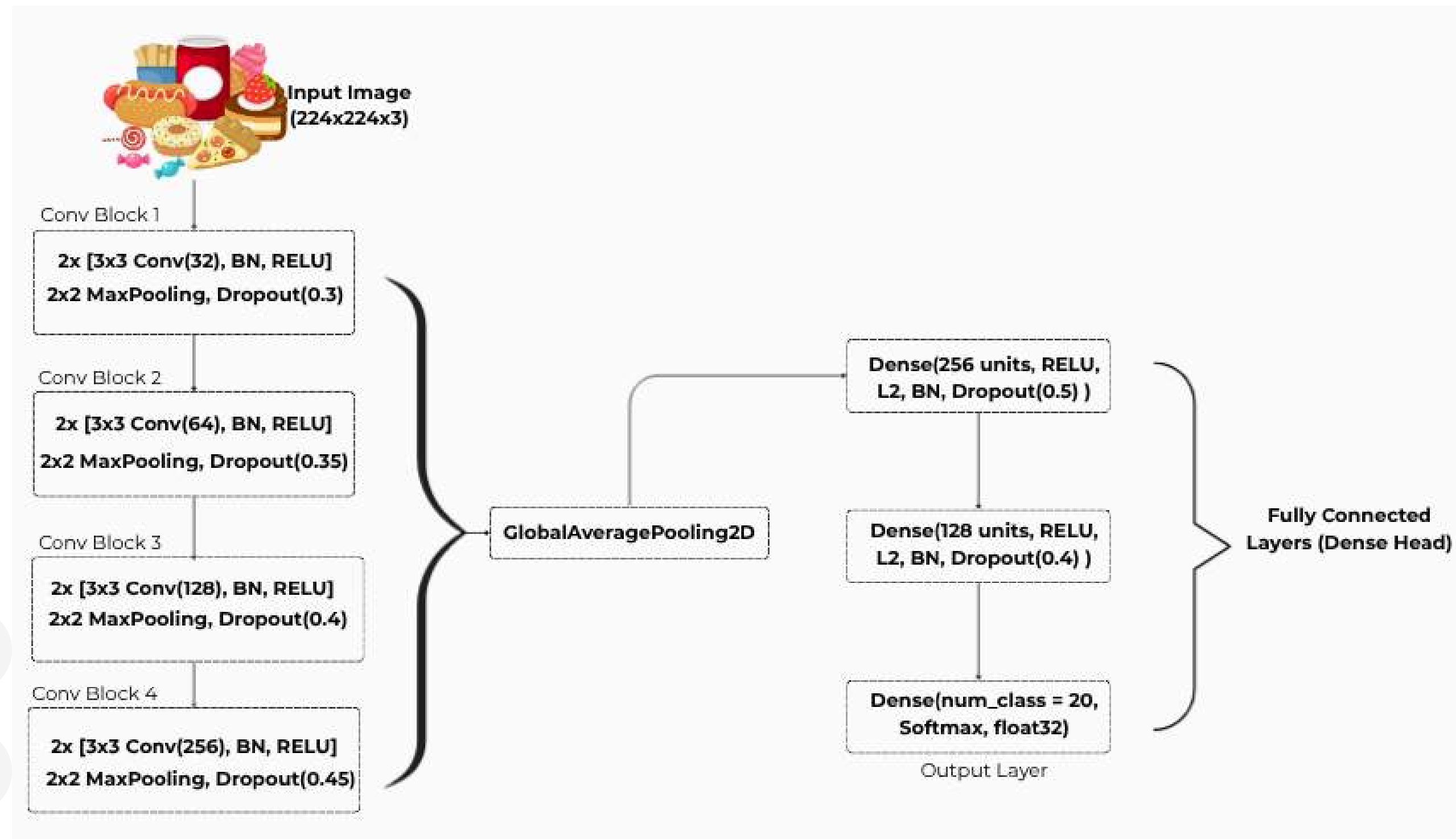
(CONV → RELU → POOL →  
DENSE → SOFTMAX)

Model Configuration Summary

NUMBER OF CLASSES

OPTIMIZER, LOSS FUNCTION

EPOCHS, BATCH SIZE



VANILLA CNN CLASSIFICATION PIPELINE

# Data Augmentation



- **Increase Data Diversity:** expose the model to a wider variety of scenarios (different angles, lighting, positions).
- **Reduce Overfitting:** makes the model more robust and prevents it from simply "memorizing" the training images, leading to better generalization on new, unseen data.

- **Geometric Changes:** Random rotation, zoom, shifting, shearing, and horizontal flipping.
- **Photometric Changes:** Varying brightness to simulate different lighting conditions.
- **Model-Specific Preprocessing:** Using `xception_preprocess` to normalize images, ensuring they are perfectly compatible with the pretrained **Xception** model.

```
train_datagen_tl = ImageDataGenerator(  
    preprocessing_function=xception_preprocess,  
    rotation_range=25,  
    width_shift_range=0.15,  
    height_shift_range=0.15,  
    shear_range=0.12,  
    zoom_range=0.2,  
    brightness_range=(0.75,1.25),  
    horizontal_flip=True,  
    fill_mode='nearest'  
)
```

# Transfer Learning

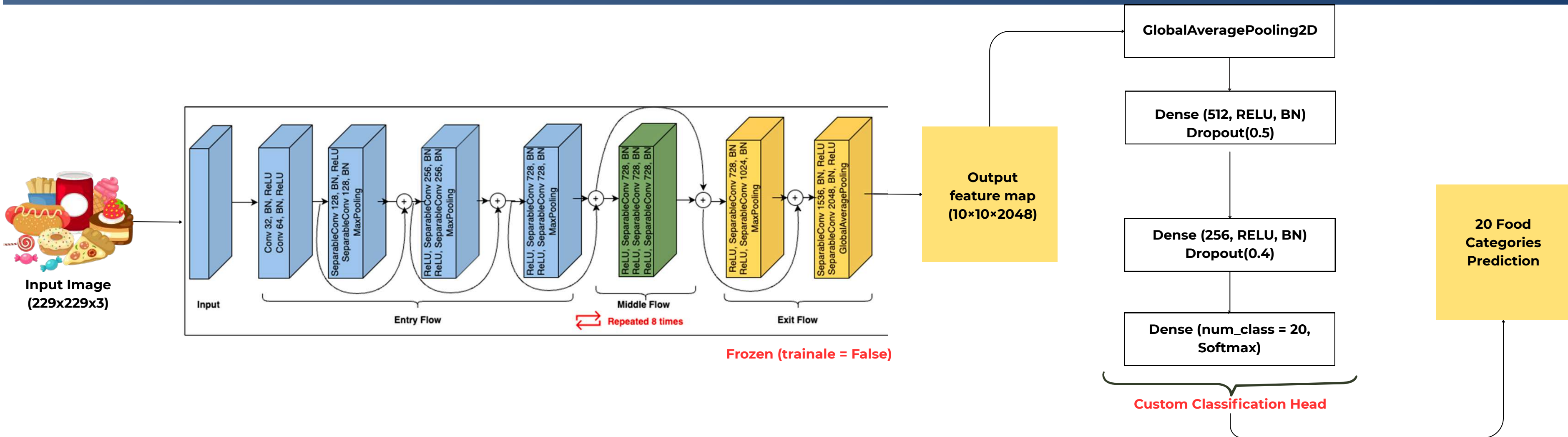
```
def build_xception_tl(input_shape=(299,299,3), num_classes=20):  
    base = Xception(weights='imagenet', include_top=False, input_shape=input_shape)  
    x = layers.GlobalAveragePooling2D()(base.output)  
    x = layers.Dense(512, activation='relu')(x)  
    x = layers.BatchNormalization()(x)  
    x = layers.Dropout(0.5)(x)  
    x = layers.Dense(256, activation='relu')(x)  
    x = layers.BatchNormalization()(x)  
    x = layers.Dropout(0.4)(x)  
    out = layers.Dense(num_classes, activation='softmax')(x)  
    model = models.Model(inputs=base.input, outputs=out)  
    return model, base  
  
model_tl, base_model = build_xception_tl((IMG_SIZE_TL, IMG_SIZE_TL, 3), NUM_CLASSES)
```

- **Why use Transfer Learning:** To leverage knowledge from pretrained models (e.g., Xception/VGG16).

## The Process:

- **Load Base:** We load the Xception model with its expert knowledge from ImageNet (weights='imagenet'), but without its original classifier head (include\_top=False).
- **Add New Head:** We stack new, trainable layers (Dense, Dropout, BatchNormalization) on top. This new "head" is built specifically to classify our food categories.
- **Combine & Freeze:** We stitch the pretrained base and new head together. The base is initially frozen to preserve its knowledge while the new head learns.

# CNN Model with Data Augmentation & Transfer Learning



Base model  
(pretrained) → Dense →  
Dropout → Output

Stage 1: Feature Extraction  
(Training the Classifier Head)

Stage 2: Fine-Tuning  
(Adapting the Top Layers)

# Stage 1: Feature Extraction (Training the Classifier Head)

```
# Train TL - stage 1
ckp_tl = ModelCheckpoint('tl_stage1_best.h5', monitor='val_loss', save_best_only=True, verbose=1)
history_tl_stage1 = model_tl.fit(
    train_gen_tl,
    epochs=6,
    validation_data=val_gen_tl,
    callbacks=[es, rlr, ckp_tl],
    verbose=1
)
```

**Model Setup:** We started with the Xception model, pretrained on ImageNet, and replaced its top layer with a new, custom classifier head (containing Dense, BatchNormalization, and Dropout layers).

**Freezing:** The entire base Xception model was frozen (`trainable = False`) to preserve its powerful, learned features.

**Initial Training:** We then trained only the new classifier head for a few epochs using a relatively standard learning rate ( $1e-4$ ). The goal was to teach the new head to classify our food images using the features extracted by the frozen base.

# Stage 2: Fine-Tuning (Adapting the Top Layers)

```
# Stage 2: unfreeze last ~100 layers and fine-tune
unfreeze_from = max(0, len(base_model.layers) - 100)
for i, layer in enumerate(base_model.layers):
    layer.trainable = True if i >= unfreeze_from else False

# (recompile with lower lr)
model_tl.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
                  loss=tf.keras.losses.CategoricalCrossentropy(label_smoothing=0.05),
                  metrics=['accuracy'])

ckp_tl2 = ModelCheckpoint('tl_finetuned_best.h5', monitor='val_loss', save_best_only=True, verbose=1)
history_tl_stage2 = model_tl.fit(
    train_gen_tl,
    epochs=30,
    validation_data=val_gen_tl,
    callbacks=[es, rlr, ckp_tl2],
    verbose=1
)
```

**Partial Unfreezing:** We unfroze the top 100 layers of the Xception model to make them trainable, while the deeper, foundational layers remained frozen to preserve their general knowledge.

**Re-compiling with a Low Learning Rate:** The model was re-compiled with a very low learning rate (1e-5). This is critical for making small, precise adjustments to the powerful pretrained weights without destroying them.

**Continued Training (Fine-Tuning):** We continued training for a larger number of epochs, allowing the now-trainable top layers to subtly adapt their specialized features to the specific details of our food dataset.

# Experimental Results

	Models	Accuracy	Precision	Recall	F1-score
1	CNN	0.8135	0.8163	0.8135	0.8123
2	CNN + Augmentation	0.845	0.8525	0.845	0.8458
3	Transfer Learning (Xception)	0.878	0.8806	0.878	0.8784
4	Transfer Learning + Augmentation	0.9025	0.9043	0.9025	0.9029

# Remarks and Analysis



## Baseline Model Limitations

While its performance was decent, the basic CNN failed to generalize and showed a tendency to overfit, highlighting the difficulty of training on a limited dataset from scratch.



## Advanced Techniques

The combination of Transfer Learning and Data Augmentation was highly effective: Transfer Learning provided robust initial features, while Data Augmentation increased data diversity to prevent overfitting.



## Performances

This combined approach achieved the highest performance across all key metrics:

- Accuracy: 90.25%
- Precision: 90.43%
- F1-Score: 90.29%

# Conclusion



The combined Transfer Learning + Augmentation model achieved the **highest accuracy** of 90.3%, decisively outperforming all baseline approaches.



- ***Transfer Learning*** provides a powerful feature foundation, especially with limited data.
- ***Data Augmentation*** is essential for reducing overfitting and improving generalization.
- The combination of both methods yields the **best** performance.



# Future Directions

- ◆ **Dataset:** Expand with more classes and real-world images to improve generalization.
- ◆ **Architectures:** Explore advanced models (e.g., EfficientNet, ViT) and techniques like ensembling.
- ◆ **Deployment:** Analyze inference speed and efficiency for mobile and edge devices.
- ◆ **Interpretability:** Use explainability tools (e.g., Grad-CAM) to understand model predictions.

**Thank You  
For Listening**

