# ITADAKI - Recipe Image Retrieval (avec Transfer Learning)

## EfficientNet (gelé) + tête personnalisée (custom head avec couches denses) pour retrouver les top-3 recettes similaires à partir d'une image

**Objectif**:

- Tester l'extraction d'embedding et la recherche par similarité avec EfficientNet gelé + transfer learning

---

## 1. Imports et configuration

In [1]:
```
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import os
import random
from PIL import Image
import re
from collections import Counter
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')

# Style dark theme
plt.style.use('dark_background')
sns.set_palette("husl")

# Seeds pour reproductibilité
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

# HYPERPARAMETERS - Recipe Image Retrieval avec EfficientNet
CONFIG = {
    # === IMAGE PARAMETERS ===
    'IMG_SIZE': 224,                    # Optimal pour EfficientNet
    'BATCH_SIZE': 32,                   # Bon équilibre mémoire/performance

    # === EMBEDDING PARAMETERS ===
    'EMBEDDING_DIM_NATIVE': 1280,       # EfficientNet features natives
    'EMBEDDING_DIM_CUSTOM': 512,        # Custom head output (si fine-tuning)

    # === PHASE 1: FEATURE EXTRACTION ===
    'USE_NATIVE_FEATURES': True,        # True = 1280-dim, False = 512-dim custom

    # === PHASE 2: TRANSFER LEARNING ===
    'TRANSFER_EPOCHS': 15,              # Entraînement head seulement
    'TRANSFER_LR': 0.001,               # Learning rate plus élevé pour nouvelles couches
    'TRANSFER_BATCH_SIZE': 32,          # Plus petit pour stabilité

    # === PHASE 3: FINE-TUNING ===
    'FINETUNE_EPOCHS': 20,              # Fine-tuning complet
    'FINETUNE_LR': 0.0001,              # Learning rate faible pour pré-entraîné
    'FINETUNE_UNFREEZE_LAYERS': 20,     # Nombre de couches à dégeler

    # === TRAINING PARAMETERS ===
    'TRIPLET_MARGIN': 0.8,              # Augmenté pour meilleure séparation
    'DROPOUT_RATE': 0.3,                # Régularisation
    'WEIGHT_DECAY': 0.0001,             # L2 regularization
```

```
    # === OPTIMIZATION ===
    'OPTIMIZER': 'adam',                # Standard
    'PATIENCE': 5,                      # Early stopping
    'REDUCE_LR_PATIENCE': 3,            # Réduction LR
    'REDUCE_LR_FACTOR': 0.5,            # Facteur réduction

    # === DATA AUGMENTATION ===
    'USE_AUGMENTATION': True,           # Pour fine-tuning
    'ROTATION_RANGE': 15,               # Rotation aléatoire
    'ZOOM_RANGE': 0.1,                  # Zoom aléatoire
    'HORIZONTAL_FLIP': True,            # Flip horizontal

    # === EVALUATION ===
    'SIMILARITY_THRESHOLD': 0.7,       # Seuil similarité
    'TOP_K_RESULTS': 3,                 # Nombre résultats retournés
    'VALIDATION_SPLIT': 0.2,           # Split pour validation
}

# HYPERPARAMÈTRES Transfer Learning
CONFIG_TL = {
    # === IMAGE PARAMETERS ===
    'IMG_SIZE': 224,                    # Optimal pour EfficientNet
    'BATCH_SIZE': 32,                   # Bon équilibre mémoire/performance

    # === EMBEDDING PARAMETERS ===
    'EMBEDDING_DIM_NATIVE': 1280,       # EfficientNet features natives
    'EMBEDDING_DIM_CUSTOM': 512,        # Custom head output

    # === TRANSFER LEARNING (TRIPLET) ===
    'TRANSFER_EPOCHS': 15,              # Entraînement avec triplet loss
    'TRANSFER_LR': 0.001,               # Learning rate pour nouvelles couches
    'TRIPLET_MARGIN': 0.8,              # Marge triplet loss
    'DROPOUT_RATE': 0.3,                # Régularisation
    'WEIGHT_DECAY': 0.0001,             # L2 regularization

    # === OPTIMIZATION AVANCÉE ===
    'OPTIMIZER': 'adam',                # Standard
    'PATIENCE': 3,                      # Early stopping
    'REDUCE_LR_PATIENCE': 2,            # Réduction LR
    'REDUCE_LR_FACTOR': 0.5,            # Facteur réduction
```

```
    'MIN_LR': 1e-7,                    # LR minimum

    # === DATA AUGMENTATION ===
    'USE_AUGMENTATION': True,          # Augmentation
    'ROTATION_RANGE': 15,              # Rotation aléatoire
    'ZOOM_RANGE': 0.1,                 # Zoom aléatoire
    'BRIGHTNESS_RANGE': [0.9, 1.1],    # Luminosité
    'HORIZONTAL_FLIP': True,           # Flip horizontal

    # === VALIDATION & ÉVALUATION ===
    'VALIDATION_SPLIT': 0.2,           # Split pour validation
    'TOP_K_RESULTS': 3,                # Nombre résultats
    'MIN_IMAGES_PER_RECIPE': 1,        # Filtrage minimum
    'MAX_PAIRS_PER_RECIPE': 5,         # Limite paires par recette

    # === MODÈLE ===
    'MODEL_NAME': 'recipe_image_retrieval_model_tl.keras',
    'EMBEDDINGS_NAME': 'recipe_embeddings_database_tl.npy',
    'METADATA_NAME': 'recipe_embeddings_database_metadata_tl.pkl'
}

print("Configuration adaptée pour EfficientNet Transfer Learning")
print(f"Phase 2: Transfer learning ({CONFIG['EMBEDDING_DIM_CUSTOM']}-dim, {CONFIG['TRANSFER_EPOCHS']} epochs)")

print("Imports et configuration OK!")
print(f"TensorFlow: {tf.__version__}")
print(f"GPU: {tf.config.list_physical_devices('GPU')}")
```

```
2025-07-14 16:45:13.620733: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-07-14 16:45:13.907284: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1752511514.021477   25288 cuda_dnn.cc:8579] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
E0000 00:00:1752511514.060250   25288 cuda_blas.cc:1407] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
W0000 00:00:1752511514.288104   25288 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1752511514.288521   25288 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1752511514.288529   25288 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1752511514.288532   25288 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
2025-07-14 16:45:14.319153: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Configuration adaptée pour EfficientNet Transfer Learning
Phase 2: Transfer learning (512-dim, 15 epochs)
Imports et configuration OK!
TensorFlow: 2.19.0
GPU: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

# 2. Définition du modèle Transfer Learning EfficientNetB0 + Custom Head

In [2]:
```python
#  TRANSFER LEARNING - Triplet Loss + EfficientNetB0
# ================================================================================
#  Triplet Loss (meilleur que Contrastive)
#  Architecture robuste + gestion d'erreurs
#  Hyperparamètres avancés (weight decay, reduce LR, validation)
#  Callbacks pour l'entraînement

import os
import pickle
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, Layer
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, TensorBoard
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.efficientnet import preprocess_input
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
import time
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')

print(f"Configuration Transfer Learning:")
print(f"- Taille des images: {CONFIG_TL['IMG_SIZE']}x{CONFIG_TL['IMG_SIZE']}")
print(f"- Taille du batch : {CONFIG_TL['BATCH_SIZE']}")
print(f"- Max Epochs: {CONFIG_TL['TRANSFER_EPOCHS']}")
print(f"- Embedding Dim: {CONFIG_TL['EMBEDDING_DIM_CUSTOM']} (custom)")
print(f"- Learning Rate: {CONFIG_TL['TRANSFER_LR']}")
print(f"- Triplet Margin: {CONFIG_TL['TRIPLET_MARGIN']}")
print(f"- Dropout Rate: {CONFIG_TL['DROPOUT_RATE']}")
print(f"- Weight Decay: {CONFIG_TL['WEIGHT_DECAY']}")
print(f"- Early Stopping: {CONFIG_TL['PATIENCE']} epochs")

# PREPROCESSING
def preprocess_image(image_path, img_size=224):
    """Preprocessing robuste avec gestion d'erreurs complète + preprocessing natif EfficientNet"""
    try:
        if not os.path.exists(image_path):
            return None

        # Vérifier la taille du fichier (éviter fichiers corrompus)
        if os.path.getsize(image_path) < 1024:  # Moins de 1KB
            return None
```

```
        img = Image.open(image_path)

        # Vérifier si l'image est valide
        img.verify()
        img = Image.open(image_path)

        # Convertir en RGB si nécessaire
        if img.mode != 'RGB':
            img = img.convert('RGB')

        # Redimensionner avec qualité optimale
        img = img.resize((img_size, img_size), Image.Resampling.LANCZOS)

        # Convertir en array et appliquer preprocessing EfficientNet
        img_array = np.array(img, dtype=np.float32)
        # Normalisation EfficientNet native
        img_array = preprocess_input(img_array)

        return img_array

    except Exception as e:
        return None

# COUCHE L2 NORMALIZATION PERSONNALISÉE (SÉRIALISABLE)
class L2NormalizationLayer(Layer):
    """Couche personnalisée pour normalisation L2 - sérialisable"""

    def __init__(self, axis=1, **kwargs):
        super(L2NormalizationLayer, self).__init__(**kwargs)
        self.axis = axis

    def call(self, inputs):
        return tf.nn.l2_normalize(inputs, axis=self.axis)

    def compute_output_shape(self, input_shape):
        return input_shape

    def get_config(self):
        config = super(L2NormalizationLayer, self).get_config()
        config.update({'axis': self.axis})
```

```
        return config

# COUCHE EXTRACTION TRIPLET PERSONNALISÉE (SÉRIALISABLE)
class ExtractTripletComponent(Layer):
    """Couche pour extraire une composante du triplet"""

    def __init__(self, component_index, **kwargs):
        super(ExtractTripletComponent, self).__init__(**kwargs)
        self.component_index = component_index

    def call(self, inputs):
        return inputs[:, self.component_index]

    def compute_output_shape(self, input_shape):
        # input_shape = (batch_size, 3, height, width, channels)
        return (input_shape[0], input_shape[2], input_shape[3], input_shape[4])

    def get_config(self):
        config = super(ExtractTripletComponent, self).get_config()
        config.update({'component_index': self.component_index})
        return config

# COUCHE STACK TRIPLET PERSONNALISÉE (SÉRIALISABLE)
class TripletStackLayer(Layer):
    """Couche personnalisée pour empiler les embeddings triplet"""

    def __init__(self, **kwargs):
        super(TripletStackLayer, self).__init__(**kwargs)

    def call(self, inputs):
        # inputs = [anchor_emb, positive_emb, negative_emb]
        return tf.stack(inputs, axis=1)

    def compute_output_shape(self, input_shape):
        # input_shape = [(batch_size, embedding_dim), ...]
        batch_size = input_shape[0][0]
        embedding_dim = input_shape[0][1]
        return (batch_size, 3, embedding_dim)

    def get_config(self):
        return super(TripletStackLayer, self).get_config()
```

```python
# ARCHITECTURE EMBEDDING
def create_embedding_model(input_shape=(224, 224, 3), embedding_dim=512):
    """Modèle d'embedding avec architecture optimisée pour triplet loss"""

    # EfficientNetB0 pré-entraîné (backbone gelé)
    base_model = EfficientNetB0(
        weights='imagenet',
        include_top=False,
        input_shape=input_shape
    )
    base_model.trainable = False  # Geler le backbone initialement

    # Pour du fine-tuning, on pourrait dégeler 20 dernières couches par exemple
    x = base_model.output

    # Custom head minimaliste
    x = base_model.output
    x = GlobalAveragePooling2D(name='global_avg_pool')(x)
    embeddings = Dense(embedding_dim, activation=None, name='embeddings')(x)

    # Normalisation L2 pour ce soit compatible avec triplet loss + similarité cosinus
    embeddings_normalized = L2NormalizationLayer(axis=1, name='l2_norm')(embeddings)
    # Modèle final
    model = Model(inputs=base_model.input, outputs=embeddings_normalized, name='TransferLearningEmbeddingModel')

    print(f"Modèle d'embedding créé:")
    print(f"- Params totaux: {model.count_params():,}")
    print(f"- Params entraînables: {sum([tf.size(w) for w in model.trainable_weights]):,}")
    print(f"- Architecture: EfficientNetB0 + Custom Head ({embedding_dim}D)")

    return model

# TRIPLET LOSS AVEC EMBEDDINGS NORMALISÉS
def triplet_loss(margin=0.3):
    """Triplet loss avec embeddings L2-normalisés"""
    def triplet_loss_fn(y_true, y_pred):
        """
        y_pred contient [anchor, positive, negative] embeddings normalisés
        Utilise distance cosinus (1 - similarity) pour la loss
        """
```

```
        anchor = y_pred[:, 0, :]      # (batch_size, embedding_dim)
        positive = y_pred[:, 1, :]    # (batch_size, embedding_dim)
        negative = y_pred[:, 2, :]    # (batch_size, embedding_dim)

        # Calcul des distances cosinus (1 - similarité cosinus)
        # Pour embeddings normalisés: cosine_sim = dot_product
        pos_similarity = tf.reduce_sum(anchor * positive, axis=1)
        neg_similarity = tf.reduce_sum(anchor * negative, axis=1)

        pos_distance = 1.0 - pos_similarity
        neg_distance = 1.0 - neg_similarity

        # Triplet loss: max(0, pos_dist - neg_dist + margin)
        basic_loss = pos_distance - neg_distance + margin
        loss = tf.maximum(0.0, basic_loss)

        return tf.reduce_mean(loss)

    return triplet_loss_fn

# visualisation des triplets
def show_triplets(generator, n_triplets=10, title=None):
    """Affiche n triplets (anchor, positive, negative) depuis un batch du générateur"""

    try:
        # Générer un batch de triplets
        triplet_batch, _ = generator._generate_triplet_batch()

        print(f"Format du batch triplet: {triplet_batch.shape}")

        # Extraire les anchors, positives, negatives
        anchors = triplet_batch[:, 0, :, :, :]
        positives = triplet_batch[:, 1, :, :, :]
        negatives = triplet_batch[:, 2, :, :, :]

        n_triplets = min(n_triplets, len(anchors))

    except Exception as e:
        print(f"Erreur lors de la génération du batch: {e}")
        return
```

```
    def normalize_image_efficientnet(img):
        """Normalise l'image EfficientNet pour l'affichage"""
        if hasattr(img, 'numpy'):
            img = img.numpy()

        # Copier pour éviter de modifier l'original
        img = np.copy(img)

        # EfficientNet preprocess_input utilise une normalisation spécifique
        # Les valeurs sont généralement dans [-1, 1] ou [-2, 2]

        # Méthode 1: Normalisation standard
        img_min = img.min()
        img_max = img.max()

        if img_max > img_min:
            img = (img - img_min) / (img_max - img_min)
        else:
            # Image uniforme, mettre à 0.5
            img = np.full_like(img, 0.5)

        return np.clip(img, 0, 1)

    # Calculer les statistiques de difficulté
    print(f"\n=== ANALYSE DE LA DIFFICULTÉ DES TRIPLETS ===")

    difficulties = []
    for i in range(min(5, n_triplets)):
        anchor = anchors[i]
        positive = positives[i]
        negative = negatives[i]

        # Calculer les distances L2
        pos_dist = np.mean((anchor - positive) ** 2)
        neg_dist = np.mean((anchor - negative) ** 2)

        difficulty_ratio = neg_dist / (pos_dist + 1e-8)
        difficulties.append(difficulty_ratio)

        print(f"Triplet {i+1}:")
        print(f"- Distance anchor-positive: {pos_dist:.4f}")
```

```
        print(f"- Distance anchor-negative: {neg_dist:.4f}")
        print(f"- Ratio difficulté: {difficulty_ratio:.2f} {'(FACILE)' if difficulty_ratio > 3 else '(DIFFICILE)'}")

    avg_difficulty = np.mean(difficulties)
    print(f"\nDifficulté moyenne: {avg_difficulty:.2f}")
    print(f"Évaluation: {'TRIPLETS TROP FACILES' if avg_difficulty > 3 else 'TRIPLETS APPROPRIÉS'}")

    # Affichage visuel
    plt.figure(figsize=(12, 4 * n_triplets))

    for i in range(n_triplets):
        for j, (img, label) in enumerate([(anchors[i], "Anchor"),
                                          (positives[i], "Positive"),
                                          (negatives[i], "Negative")]):
            plt.subplot(n_triplets, 3, i * 3 + j + 1)

            try:
                # Utiliser la nouvelle fonction de normalisation
                normalized_img = normalize_image_efficientnet(img)

                # Debug: afficher les stats de l'image
                print(f"Image {i+1}-{label}: min={img.min():.3f}, max={img.max():.3f}, mean={img.mean():.3f}")
                print(f"Normalisée: min={normalized_img.min():.3f}, max={normalized_img.max():.3f}")

                plt.imshow(normalized_img)
                plt.axis('off')
                plt.title(f"{label}")

                # Ajouter info de difficulté pour le premier triplet
                if i == 0 and j == 0:
                    plt.xlabel(f"Difficulté: {difficulties[0]:.2f}")

            except Exception as e:
                print(f"Erreur affichage image {i+1}-{label}: {e}")
                plt.text(0.5, 0.5, f"Erreur: {str(e)[:50]}",
                        ha='center', va='center', transform=plt.gca().transAxes)
                plt.axis('off')

    if title:
        plt.suptitle(title, fontsize=16)
    else:
```

```
        plt.suptitle(f"Triplets - Difficulté moyenne: {avg_difficulty:.2f}", fontsize=16)

    plt.tight_layout()
    plt.show()

# GÉNÉRATEUR DE TRIPLETS
class TripletGenerator(tf.keras.utils.Sequence):
    """Générateur de triplets avec validation et augmentation avancée"""

    def __init__(self, recipes_df, batch_size=32, img_size=224, augment=True, validation_split=0.2, is_validation=False):
        self.batch_size = batch_size
        self.img_size = img_size
        self.augment = augment and not is_validation
        self.is_validation = is_validation

        # Filtrer et diviser les données
        self.valid_recipes = self._filter_valid_recipes(recipes_df)
        self.train_recipes, self.val_recipes = self._split_data(validation_split)

        # Utiliser le bon subset
        self.recipes_df = self.val_recipes if is_validation else self.train_recipes

        # Grouper par titre de recette
        self.recipe_groups = self.recipes_df.groupby('Title')
        self.recipe_names = list(self.recipe_groups.groups.keys())

        # Configuration augmentation de données
        if self.augment:
            self.datagen = ImageDataGenerator(
                rotation_range=CONFIG_TL['ROTATION_RANGE'],
                width_shift_range=0.1,
                height_shift_range=0.1,
                horizontal_flip=CONFIG_TL['HORIZONTAL_FLIP'],
                zoom_range=CONFIG_TL['ZOOM_RANGE'],
                brightness_range=CONFIG_TL['BRIGHTNESS_RANGE'],
                fill_mode='nearest'
            )

        self.on_epoch_end()

        split_name = "Validation" if is_validation else "Entraînement"
```

```
        print(f"Générateur {split_name} créé:")
        print(f"- Recettes: {len(self.recipe_names)}")
        print(f"- Images: {len(self.recipes_df)}")
        print(f"- Augmentation: {'Activée' if self.augment else 'Désactivée'}")

    def _filter_valid_recipes(self, recipes_df):
        """Filtrer les recettes avec images valides"""
        valid_recipes = []

        print("Filtrage des images valides...")
        for idx, row in tqdm(recipes_df.iterrows(), total=len(recipes_df), desc="Validation images"):
            if pd.notna(row['image_path']) and os.path.exists(row['image_path']):
                try:
                    img = Image.open(row['image_path'])
                    img.verify()
                    valid_recipes.append(row)
                except:
                    continue

        result_df = pd.DataFrame(valid_recipes)
        print(f"{len(result_df)}/{len(recipes_df)} images valides")
        return result_df

    def _split_data(self, validation_split):
        """Diviser les données en train/validation par recette"""
        unique_recipes = self.valid_recipes['Title'].unique()

        train_recipes, val_recipes = train_test_split(
            unique_recipes,
            test_size=validation_split,
            random_state=42
        )

        train_df = self.valid_recipes[self.valid_recipes['Title'].isin(train_recipes)]
        val_df = self.valid_recipes[self.valid_recipes['Title'].isin(val_recipes)]

        return train_df, val_df

    def __len__(self):
        return max(1, len(self.recipes_df) // self.batch_size)
```

```
    def __getitem__(self, index):
        return self._generate_triplet_batch()

    def on_epoch_end(self):
        """Mélanger les données à chaque époque"""
        pass

    def _generate_triplet_batch(self):
        """Générer un batch de triplets avec gestion robuste des erreurs"""
        batch_size = self.batch_size

        anchors = np.zeros((batch_size, self.img_size, self.img_size, 3), dtype=np.float32)
        positives = np.zeros((batch_size, self.img_size, self.img_size, 3), dtype=np.float32)
        negatives = np.zeros((batch_size, self.img_size, self.img_size, 3), dtype=np.float32)

        valid_triplets = 0
        attempts = 0
        # Plus de tentatives pour robustesse
        max_attempts = batch_size * 5

        while valid_triplets < batch_size and attempts < max_attempts:
            attempts += 1

            try:
                # Sélectionner une recette positive (avec au moins 1 image)
                pos_recipe = np.random.choice(self.recipe_names)
                pos_group = self.recipe_groups.get_group(pos_recipe)

                # Sélectionner une recette négative différente
                available_neg_recipes = [r for r in self.recipe_names if r != pos_recipe]
                if not available_neg_recipes:
                    continue

                neg_recipe = np.random.choice(available_neg_recipes)
                neg_group = self.recipe_groups.get_group(neg_recipe)

                # Sélectionner les images du triplet
                if len(pos_group) >= 2:
                    # Cas idéal: 2 images différentes de la même recette
                    anchor_row, pos_row = pos_group.sample(2).iloc
                else:
```

```
                # Cas limite: même image pour anchor et positive
                anchor_row = pos_row = pos_group.iloc[0]

            # Image négative d'une recette différente
            neg_row = neg_group.sample(1).iloc[0]

            # Charger et préprocesser les images
            anchor_img = preprocess_image(anchor_row['image_path'], self.img_size)
            pos_img = preprocess_image(pos_row['image_path'], self.img_size)
            neg_img = preprocess_image(neg_row['image_path'], self.img_size)

            # Vérifier que toutes les images sont valides
            if anchor_img is not None and pos_img is not None and neg_img is not None:

                # Appliquer l'augmentation de données
                if self.augment:
                    # Pas d'augmentation sur anchor et negative
                    pos_img = self.datagen.random_transform(pos_img)


                # Ajouter au batch
                anchors[valid_triplets] = anchor_img
                positives[valid_triplets] = pos_img
                negatives[valid_triplets] = neg_img

                valid_triplets += 1

        except Exception as e:
            print(f"Erreur sur un triplet: {e}")
            continue

    # Compléter avec des copies si nécessaire (cas rare)
    while valid_triplets < batch_size:
        print(f"Pas assez de triplets valides: {valid_triplets}")
        copy_idx = np.random.randint(0, max(1, valid_triplets))
        anchors[valid_triplets] = anchors[copy_idx]
        positives[valid_triplets] = positives[copy_idx]
        negatives[valid_triplets] = negatives[copy_idx]
        valid_triplets += 1

    # Format pour le modèle triplet: (batch_size, 3, height, width, channels)
```

```
        triplet_batch = np.stack([anchors, positives, negatives], axis=1)
        dummy_labels = np.zeros((batch_size, 1))  # Non utilisé avec triplet loss

        return triplet_batch, dummy_labels

class HardTripletGenerator(TripletGenerator):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.embedding_cache = {}
        self.update_frequency = 50  # Recalculer tous les 50 batches
        self.batch_count = 0

    def _select_hard_negative(self, pos_recipe, anchor_row):
        """Sélectionne un negative difficile"""
        # Pour l'instant, sélection aléatoire (à améliorer avec embeddings)
        available_neg_recipes = [r for r in self.recipe_names if r != pos_recipe]

        # TODO: Améliorer avec similarité des embeddings
        # Pour l'instant, on fait du random
        return np.random.choice(available_neg_recipes)

    def _strong_augmentation(self, img):
        """Augmentation plus forte pour rendre les positives plus difficiles"""
        # Attention: img est déjà normalisé par EfficientNet preprocess_input
        # Les valeurs sont dans [-1, 1] environ

        # Augmentation normale d'abord
        img = self.datagen.random_transform(img)

        # Augmentations supplémentaires légères
        if np.random.random() < 0.3:
            # Bruit gaussien très léger
            noise = np.random.normal(0, 0.02, img.shape)
            img = img + noise

        if np.random.random() < 0.2:
            # Légère modification de contraste
            img = img * np.random.uniform(0.95, 1.05)

        return img
```

```
    def _generate_triplet_batch(self):
        """Version avec hard negative mining"""
        batch_size = self.batch_size

        anchors = np.zeros((batch_size, self.img_size, self.img_size, 3), dtype=np.float32)
        positives = np.zeros((batch_size, self.img_size, self.img_size, 3), dtype=np.float32)
        negatives = np.zeros((batch_size, self.img_size, self.img_size, 3), dtype=np.float32)

        valid_triplets = 0
        attempts = 0
        max_attempts = batch_size * 5

        while valid_triplets < batch_size and attempts < max_attempts:
            attempts += 1

            try:
                # Sélection anchor/positive (même logique)
                pos_recipe = np.random.choice(self.recipe_names)
                pos_group = self.recipe_groups.get_group(pos_recipe)

                if len(pos_group) >= 2:
                    anchor_row, pos_row = pos_group.sample(2).iloc
                else:
                    anchor_row = pos_row = pos_group.iloc[0]

                # Hard negative selection
                neg_recipe = self._select_hard_negative(pos_recipe, anchor_row)
                neg_group = self.recipe_groups.get_group(neg_recipe)
                neg_row = neg_group.sample(1).iloc[0]

                # Charger les images
                anchor_img = preprocess_image(anchor_row['image_path'], self.img_size)
                pos_img = preprocess_image(pos_row['image_path'], self.img_size)
                neg_img = preprocess_image(neg_row['image_path'], self.img_size)

                if anchor_img is not None and pos_img is not None and neg_img is not None:

                    # Augmentation PLUS FORTE sur positive
                    if self.augment:
                        pos_img = self._strong_augmentation(pos_img)
```

```
                    anchors[valid_triplets] = anchor_img
                    positives[valid_triplets] = pos_img
                    negatives[valid_triplets] = neg_img

                    valid_triplets += 1

            except Exception as e:
                print(f"Erreur triplet: {e}")
                continue

        # Compléter si nécessaire
        while valid_triplets < batch_size:
            copy_idx = np.random.randint(0, max(1, valid_triplets))
            anchors[valid_triplets] = anchors[copy_idx]
            positives[valid_triplets] = positives[copy_idx]
            negatives[valid_triplets] = negatives[copy_idx]
            valid_triplets += 1

        triplet_batch = np.stack([anchors, positives, negatives], axis=1)
        dummy_labels = np.zeros((batch_size, 1))

        return triplet_batch, dummy_labels

# MODÈLE TRIPLET POUR ENTRAÎNEMENT
def create_triplet_model(embedding_model):

    """Modèle triplet avec couches personnalisées sérialisables"""

    # Input: batch de triplets (batch_size, 3, height, width, channels)
    triplet_input = tf.keras.layers.Input(
        shape=(3, CONFIG_TL['IMG_SIZE'], CONFIG_TL['IMG_SIZE'], 3),
        name='triplet_input'
    )

    anchor = ExtractTripletComponent(0, name='extract_anchor')(triplet_input)
    positive = ExtractTripletComponent(1, name='extract_positive')(triplet_input)
    negative = ExtractTripletComponent(2, name='extract_negative')(triplet_input)

    # Obtenir les embeddings normalisés
    anchor_emb = embedding_model(anchor)
    positive_emb = embedding_model(positive)
```

```
    negative_emb = embedding_model(negative)

    triplet_output = TripletStackLayer(name='combine_embeddings')([anchor_emb, positive_emb, negative_emb])

    model = Model(inputs=triplet_input, outputs=triplet_output, name='TransferLearningTripletModel')

    print(f"Modèle triplet créé:")
    print(f"- Input shape: {model.input_shape}")
    print(f"- Output shape: {model.output_shape}")

    return model

# MÉTRIQUES
def triplet_accuracy(y_true, y_pred):
    """Précision: % de triplets où positive plus proche que negative"""
    anchor = y_pred[:, 0, :]     # (batch_size, embedding_dim)
    positive = y_pred[:, 1, :]   # (batch_size, embedding_dim)
    negative = y_pred[:, 2, :]   # (batch_size, embedding_dim)

    # Similarités cosinus (plus élevé = plus similaire)
    pos_similarity = tf.reduce_sum(anchor * positive, axis=1)
    neg_similarity = tf.reduce_sum(anchor * negative, axis=1)

    # Triplet correct si positive plus similaire que negative
    correct_triplets = tf.cast(pos_similarity > neg_similarity, tf.float32)
    accuracy = tf.reduce_mean(correct_triplets)

    return accuracy

def triplet_margin_accuracy(y_true, y_pred):
    """% de triplets qui respectent la marge complète (vraie réussite)"""
    anchor = y_pred[:, 0, :]
    positive = y_pred[:, 1, :]
    negative = y_pred[:, 2, :]

    pos_similarity = tf.reduce_sum(anchor * positive, axis=1)
    neg_similarity = tf.reduce_sum(anchor * negative, axis=1)

    # Distance cosinus
    pos_distance = 1.0 - pos_similarity
    neg_distance = 1.0 - neg_similarity
```

```
    # Triplet satisfait si : neg_dist - pos_dist > margin
    margin = CONFIG_TL['TRIPLET_MARGIN']
    margin_satisfied = tf.cast((neg_distance - pos_distance) > margin, tf.float32)

    return tf.reduce_mean(margin_satisfied)

def average_positive_similarity(y_true, y_pred):
    """Similarité moyenne anchor-positive"""
    anchor = y_pred[:, 0, :]
    positive = y_pred[:, 1, :]
    pos_similarity = tf.reduce_sum(anchor * positive, axis=1)
    return tf.reduce_mean(pos_similarity)

def average_negative_similarity(y_true, y_pred):
    """Similarité moyenne anchor-negative"""
    anchor = y_pred[:, 0, :]
    negative = y_pred[:, 2, :]
    neg_similarity = tf.reduce_sum(anchor * negative, axis=1)
    return tf.reduce_mean(neg_similarity)

# FONCTION D'ENTRAÎNEMENT PRINCIPALE
def train_transfer_learning():
    """Entraînement complet avec toutes les optimisations"""

    print("Début de l'entraînement Transfer Learning...")
    print("=" * 60)

    # Charger les données
    try:
        recipes_df = pd.read_pickle("./data/recipes_with_images_dataframe.pkl")
        print(f"{len(recipes_df)} recettes chargées depuis le DataFrame")
    except FileNotFoundError:
        print("Fichier ./data/recipes_with_images_dataframe.pkl introuvable")
        print("Assurez-vous d'avoir exécuté le notebook 'raw' qui génère le DataFrame sous forme de pickle")
        return None, None

    # Créer le modèle d'embedding
    print("\nCréation du modèle d'embedding...")
    embedding_model = create_embedding_model(
        input_shape=(CONFIG_TL['IMG_SIZE'], CONFIG_TL['IMG_SIZE'], 3),
```

```
    embedding_dim=CONFIG_TL['EMBEDDING_DIM_CUSTOM']
)

# Créer les générateurs de données (train + validation)
print("\nCréation des générateurs de données...")
train_generator = HardTripletGenerator(
    recipes_df,
    batch_size=CONFIG_TL['BATCH_SIZE'],
    img_size=CONFIG_TL['IMG_SIZE'],
    augment=CONFIG_TL['USE_AUGMENTATION'],
    validation_split=CONFIG_TL['VALIDATION_SPLIT'],
    is_validation=False
)

show_triplets(train_generator, n_triplets=10, title="Triplets train")

val_generator = HardTripletGenerator(
    recipes_df,
    batch_size=CONFIG_TL['BATCH_SIZE'],
    img_size=CONFIG_TL['IMG_SIZE'],
    augment=False,  # Pas d'augmentation en validation
    validation_split=CONFIG_TL['VALIDATION_SPLIT'],
    is_validation=True
)

show_triplets(val_generator, n_triplets=10, title="Triplets val")

# Créer le modèle triplet
triplet_model = create_triplet_model(embedding_model)

# Compiler le modèle avec optimisations
triplet_model.compile(
    optimizer=Adam(
        learning_rate=CONFIG_TL['TRANSFER_LR'],
        weight_decay=CONFIG_TL['WEIGHT_DECAY']
    ),
    loss=triplet_loss(margin=CONFIG_TL['TRIPLET_MARGIN']),
    metrics=[triplet_margin_accuracy, average_positive_similarity, average_negative_similarity]
)

print(f"Modèle compilé:")
```

```
print(f"- Optimizer: Adam (lr={CONFIG_TL['TRANSFER_LR']}, wd={CONFIG_TL['WEIGHT_DECAY']})")
print(f"- Loss: Triplet Loss (margin={CONFIG_TL['TRIPLET_MARGIN']})")
print(f"- Métriques: Accuracy + Similarités moyenne")

# CALLBACK PERSONNALISÉ pour adapter le margin de triplet loss
class AdaptiveMarginCallback(tf.keras.callbacks.Callback):
    def __init__(self, initial_margin=CONFIG_TL['TRIPLET_MARGIN']):
        self.margin = initial_margin

    def on_epoch_end(self, epoch, logs=None):
        triplet_acc = logs.get('triplet_margin_accuracy', 0)

        if triplet_acc > 0.95:  # Trop facile
            self.margin += 0.1
            print(f"Augmentation margin à {self.margin}")
        elif triplet_acc < 0.7:  # Trop difficile
            self.margin -= 0.05
            print(f"Réduction margin à {self.margin}")

# CALLBACK PERSONNALISÉ pour sauvegarder UNIQUEMENT l'embedding model (et pas le triplet model)
class EmbeddingModelCheckpoint(tf.keras.callbacks.Callback):
    """
    Callback personnalisé qui sauvegarde l'embedding model (pas le triplet model)
    C'est CRUCIAL car ModelCheckpoint standard sauvegarderait le triplet model !
    """
    def __init__(self, embedding_model, filepath, monitor='val_loss', save_best_only=True, verbose=1):
        super().__init__()
        self.embedding_model = embedding_model  # Le modèle qui nous intéresse
        self.filepath = filepath
        self.monitor = monitor
        self.save_best_only = save_best_only
        self.verbose = verbose
        self.best_value = np.inf if 'loss' in monitor else -np.inf

    def on_epoch_end(self, epoch, logs=None):
        current_value = logs.get(self.monitor)
        if current_value is None:
            if self.verbose > 0:
                print(f"\n Métrique '{self.monitor}' non trouvée dans les logs")
            return
```

```
            improved = False
            if 'loss' in self.monitor or 'error' in self.monitor:
                # Pour les métriques à minimiser
                if current_value < self.best_value:
                    self.best_value = current_value
                    improved = True
            else:
                # Pour les métriques à maximiser (accuracy, etc.)
                if current_value > self.best_value:
                    self.best_value = current_value
                    improved = True

            if improved or not self.save_best_only:
                try:
                    #  SAUVEGARDER L'EMBEDDING MODEL si amélioration (pas le triplet model)
                    if improved:
                        self.embedding_model.save(self.filepath)
                        print(f"\nNOUVEAU MEILLEUR MODELE D'EMBEDDINGS sauvegardé!")
                        print(f"Fichier: {self.filepath}")
                        print(f"{self.monitor}: {current_value:.4f}")
                    else:
                        print(f"\nPas d'amélioration du modèle d'embeddings")

                except Exception as e:
                    if self.verbose > 0:
                        print(f"\nErreur sauvegarde embedding model: {e}")

        def on_train_end(self, logs=None):
            if self.verbose > 0:
                print(f"\nEntraînement terminé. Meilleur {self.monitor}: {self.best_value:.4f}")
                print(f"Meilleur embedding model disponible: {self.filepath}")

    # Callbacks pour l'entraînement
    callbacks = [
        EarlyStopping(
            monitor='val_loss',
            patience=CONFIG_TL['PATIENCE'],
            restore_best_weights=True,
            verbose=1,
            mode='min'
        ),
```

```
        ReduceLROnPlateau(
            monitor='val_loss',
            patience=CONFIG_TL['REDUCE_LR_PATIENCE'],
            factor=CONFIG_TL['REDUCE_LR_FACTOR'],
            min_lr=CONFIG_TL['MIN_LR'],
            verbose=1,
            mode='min'
        ),
        ModelCheckpoint(
            f"./tl/best_triplet_{CONFIG_TL['MODEL_NAME']}",
            monitor='val_loss',
            save_best_only=True,
            save_weights_only=False,
            verbose=1,
            mode='min'
        ),
        AdaptiveMarginCallback(initial_margin=CONFIG_TL['TRIPLET_MARGIN']),
        EmbeddingModelCheckpoint(
            embedding_model,
            f"./tl/best_embedding_{CONFIG_TL['MODEL_NAME']}",
            monitor='val_loss',
            save_best_only=True,
            verbose=1,
        ),
        TensorBoard(
            log_dir=f"./tl/logs/{CONFIG_TL['MODEL_NAME']}",
            histogram_freq=1,
            write_graph=True,
            write_images=True,
            update_freq='epoch',
            profile_batch=2,
            embeddings_freq=1
        )
    ]

    print(f"Callbacks configurés:")
    print(f"- Adaptive Margin: initial={CONFIG_TL['TRIPLET_MARGIN']}")
    print(f"- Early Stopping: patience={CONFIG_TL['PATIENCE']} (val_loss)")
    print(f"- Reduce LR: patience={CONFIG_TL['REDUCE_LR_PATIENCE']}, factor={CONFIG_TL['REDUCE_LR_FACTOR']}")
    print(f"- Model Checkpoint: best_{CONFIG_TL['MODEL_NAME']}")
```

```
    # Entraînement
    print("\n" + "="*60)
    print("DÉBUT DE L'ENTRAÎNEMENT")
    print("="*60)
    print(f"- Entraînement: {len(train_generator)} batch/époque")
    print(f"- Validation: {len(val_generator)} batch/époque")
    print(f"- Objectif: apprendre des embeddings discriminants via triplet loss")

    start_time = time.time()

    history = triplet_model.fit(
        train_generator,
        epochs=CONFIG_TL['TRANSFER_EPOCHS'],
        validation_data=val_generator,
        callbacks=callbacks,
        verbose=1
    )

    training_time = time.time() - start_time

    # Afficher les résultats
    print("\n" + "="*60)
    print(" ENTRAÎNEMENT TERMINÉ AVEC SUCCÈS!")
    print("="*60)
    print(f"⏱️ Temps total: {training_time/60:.1f} minutes")
    print(f"- Meilleure val_loss: {min(history.history['val_loss']):.4f}")
    print(f"- Précision finale: {history.history['val_triplet_margin_accuracy'][-1]:.4f}")
    print(f"- Similarité pos finale: {history.history['val_average_positive_similarity'][-1]:.4f}")
    print(f"- Similarité neg finale: {history.history['val_average_negative_similarity'][-1]:.4f}")
    print(f"- Modèle sauvegardé: {CONFIG_TL['MODEL_NAME']}")

    return embedding_model, history

#  VISUALISATION DES RÉSULTATS
def plot_training_results(history):
    """Afficher les métriques d'entraînement avec échelles adaptées"""

    # Vérifier que les métriques existent
    required_metrics = ['loss', 'val_loss', 'triplet_margin_accuracy', 'val_triplet_margin_accuracy']
    for metric in required_metrics:
        if metric not in history.history:
```

```
        print(f"Métrique manquante: {metric}")
        return

# Configuration dark mode (identique)
plt.style.use('dark_background')
colors = {
    'primary': '#00D4AA',
    'secondary': '#FF6B6B',
    'accent': '#4ECDC4',
    'warning': '#FFE66D',
    'info': '#A8E6CF',
    'purple': '#B19CD9'
}

fig, axes = plt.subplots(2, 2, figsize=(16, 12))
fig.patch.set_facecolor('#1e1e1e')
fig.suptitle('Résultats d\'Entraînement Fine-Tuning',
             fontsize=18, fontweight='bold', color='white', y=0.95)

for ax in axes.flat:
    ax.set_facecolor('#2d2d2d')
    ax.tick_params(colors='white', which='both')
    ax.xaxis.label.set_color('white')
    ax.yaxis.label.set_color('white')
    ax.title.set_color('white')
    ax.grid(True, alpha=0.2, color='gray', linestyle='-', linewidth=0.5)
    for spine in ax.spines.values():
        spine.set_color('white')

# Loss
axes[0, 0].plot(history.history['loss'], label='Train Loss',
                linewidth=3, color=colors['primary'], alpha=0.9)
axes[0, 0].plot(history.history['val_loss'], label='Val Loss',
                linewidth=3, color=colors['secondary'], alpha=0.9)
axes[0, 0].fill_between(range(len(history.history['loss'])),
                        history.history['loss'], alpha=0.1, color=colors['primary'])
axes[0, 0].fill_between(range(len(history.history['val_loss'])),
                        history.history['val_loss'], alpha=0.1, color=colors['secondary'])
axes[0, 0].set_title('Triplet Loss Evolution', fontweight='bold', fontsize=14)
axes[0, 0].set_xlabel('Époque', fontweight='bold')
axes[0, 0].set_ylabel('Loss Value', fontweight='bold')
```

```
    axes[0, 0].legend(frameon=True, fancybox=True, shadow=True,
                      facecolor='#3d3d3d', edgecolor='white')

    # Accuracy
    axes[0, 1].plot(history.history['triplet_margin_accuracy'], label='Train Margin Accuracy',
                    linewidth=3, color=colors['accent'], alpha=0.9)
    axes[0, 1].plot(history.history['val_triplet_margin_accuracy'], label='Val Margin Accuracy',
                    linewidth=3, color=colors['warning'], alpha=0.9)
    axes[0, 1].axhline(y=0.8, color='white', linestyle='--', alpha=0.5, label='Target (80%)')
    axes[0, 1].set_title('Triplet Accuracy Progress', fontweight='bold', fontsize=14)
    axes[0, 1].set_xlabel('Époque', fontweight='bold')
    axes[0, 1].set_ylabel('Accuracy Score', fontweight='bold')
    axes[0, 1].legend(frameon=True, fancybox=True, shadow=True,
                      facecolor='#3d3d3d', edgecolor='white')
    axes[0, 1].set_ylim(0, 1)  # ✅ Parfait

    # Positive Similarity
    if 'average_positive_similarity' in history.history:
        axes[1, 0].plot(history.history['average_positive_similarity'], label='Train Pos Sim',
                        linewidth=3, color=colors['info'], alpha=0.9)
        axes[1, 0].plot(history.history['val_average_positive_similarity'], label='Val Pos Sim',
                        linewidth=3, color=colors['purple'], alpha=0.9)


        axes[1, 0].set_ylim(0, 1)
        axes[1, 0].axhspan(0.7, 1.0, alpha=0.1, color='green', label='Target Zone (>0.7)')
        axes[1, 0].axhline(y=0.7, color='white', linestyle='--', alpha=0.5)

    axes[1, 0].set_title('Positive Similarity (Anchor-Positive)', fontweight='bold', fontsize=14)
    axes[1, 0].set_xlabel('Époque', fontweight='bold')
    axes[1, 0].set_ylabel('Cosine Similarity', fontweight='bold')
    axes[1, 0].legend(frameon=True, fancybox=True, shadow=True,
                      facecolor='#3d3d3d', edgecolor='white')

    # Negative Similarity
    if 'average_negative_similarity' in history.history:
        axes[1, 1].plot(history.history['average_negative_similarity'], label='Train Neg Sim',
                        linewidth=3, color=colors['secondary'], alpha=0.9)
        axes[1, 1].plot(history.history['val_average_negative_similarity'], label='Val Neg Sim',
                        linewidth=3, color=colors['primary'], alpha=0.9)
```

```
    axes[1, 1].set_ylim(-0.5, 0.5)
    axes[1, 1].axhspan(-0.5, 0.3, alpha=0.1, color='red', label='Target Zone (<0.3)')
    axes[1, 1].axhline(y=0.3, color='white', linestyle='--', alpha=0.5)

axes[1, 1].set_title('Negative Similarity (Anchor-Negative)', fontweight='bold', fontsize=14)
axes[1, 1].set_xlabel('Époque', fontweight='bold')
axes[1, 1].set_ylabel('Cosine Similarity', fontweight='bold')
axes[1, 1].legend(frameon=True, fancybox=True, shadow=True,
                  facecolor='#3d3d3d', edgecolor='white')

plt.tight_layout()
plt.subplots_adjust(top=0.92)
plt.show()

plt.style.use('default')

# Résumé amélioré
print("\nRÉSUMÉ DES MÉTRIQUES FINALES:")
print("-" * 50)

final_metrics = {}
if 'val_loss' in history.history:
    final_metrics['Loss (validation)'] = history.history['val_loss'][-1]
if 'val_triplet_margin_accuracy' in history.history:
    final_metrics['Margin Accuracy (validation)'] = history.history['val_triplet_margin_accuracy'][-1]
if 'val_average_positive_similarity' in history.history:
    final_metrics['Similarité Positive'] = history.history['val_average_positive_similarity'][-1]
if 'val_average_negative_similarity' in history.history:
    final_metrics['Similarité Négative'] = history.history['val_average_negative_similarity'][-1]

# Calcul de l'écart seulement si les deux métriques existent
if 'val_average_positive_similarity' in history.history and 'val_average_negative_similarity' in history.history:
    final_metrics['Écart Pos-Neg'] = (history.history['val_average_positive_similarity'][-1] -
                                      history.history['val_average_negative_similarity'][-1])

for metric, value in final_metrics.items():
    print(f"{metric:25} : {value:.4f}")

# Interprétation
print("\nINTERPRÉTATION:")
if 'val_triplet_margin_accuracy' in history.history:
```

```
        acc = history.history['val_triplet_margin_accuracy'][-1]
        if acc > 0.85:
            print(f"✅ Excellente accuracy ({acc:.1%})")
        elif acc > 0.75:
            print(f"🟡 Bonne accuracy ({acc:.1%})")
        else:
            print(f"❌ Accuracy faible ({acc:.1%}) - modèle à améliorer")
```

```
Configuration Transfer Learning:
- Taille des images: 224x224
- Taille du batch : 32
- Max Epochs: 15
- Embedding Dim: 512 (custom)
- Learning Rate: 0.001
- Triplet Margin: 0.8
- Dropout Rate: 0.3
- Weight Decay: 0.0001
- Early Stopping: 3 epochs
```

## 3. Entraînement du modèle

In [3]:
```
# Entraîner le modèle
embedding_model, history = train_transfer_learning()

if embedding_model is not None and history is not None:
    # Afficher les résultats
    plot_training_results(history)

    print("\nSUCCÈS! Entraînement terminé.")
    print(f"Modèle prêt: {CONFIG_TL['MODEL_NAME']}")
    print("Prêt pour l'extraction d'embeddings et la recherche par similarité!")

else:
    print("Échec de l'entraînement. Vérifiez les données et ressayez.")
```

```
Début de l'entraînement Transfer Learning...
============================================================
13463 recettes chargées depuis le DataFrame

Création du modèle d'embedding...
```

```
I0000 00:00:1752511520.319666   25288 gpu_device.cc:2019] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 2053
2 MB memory:  -> device: 0, name: NVIDIA L4, pci bus id: 0000:00:04.0, compute capability: 8.9
Modèle d'embedding créé:
- Params totaux: 4,705,443
- Params entraînables: 655,872
- Architecture: EfficientNetB0 + Custom Head (512D)

Création des générateurs de données...
Filtrage des images valides...
Validation images: 100%|██████████| 13463/13463 [00:03<00:00, 3391.08it/s]
```

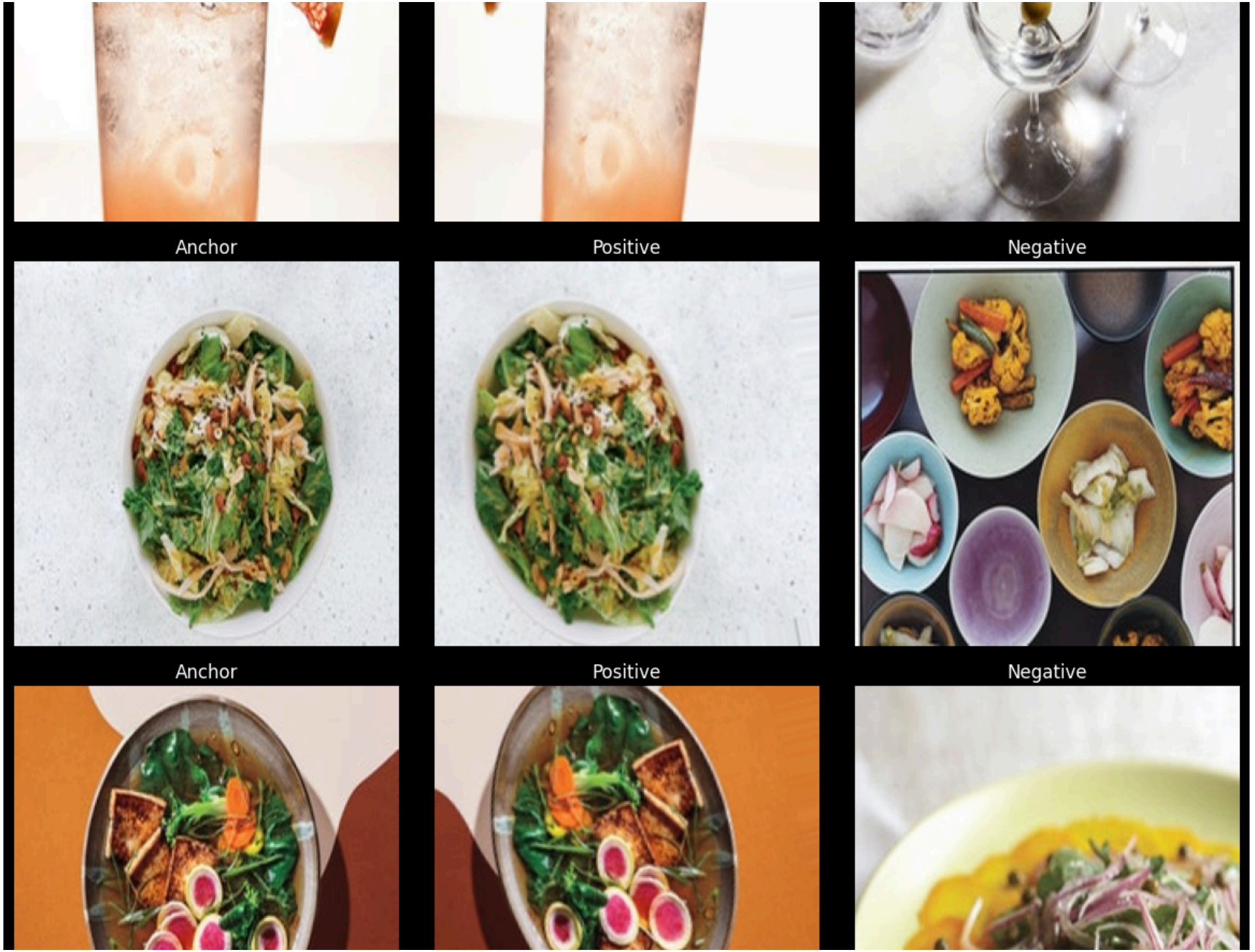
```
13463/13463 images valides
Générateur Entraînement créé:
- Recettes: 10618
- Images: 10774
- Augmentation: Activée
Format du batch triplet: (32, 3, 224, 224, 3)

=== ANALYSE DE LA DIFFICULTÉ DES TRIPLETS ===
Triplet 1:
- Distance anchor-positive: 1746.1373
- Distance anchor-negative: 11155.1191
- Ratio difficulté: 6.39 (FACILE)
Triplet 2:
- Distance anchor-positive: 7082.8530
- Distance anchor-negative: 11876.2480
- Ratio difficulté: 1.68 (DIFFICILE)
Triplet 3:
- Distance anchor-positive: 2813.7075
- Distance anchor-negative: 8177.8477
- Ratio difficulté: 2.91 (DIFFICILE)
Triplet 4:
- Distance anchor-positive: 11532.9121
- Distance anchor-negative: 32249.8809
- Ratio difficulté: 2.80 (DIFFICILE)
Triplet 5:
- Distance anchor-positive: 3860.3262
- Distance anchor-negative: 4879.7715
- Ratio difficulté: 1.26 (DIFFICILE)

Difficulté moyenne: 3.01
Évaluation: TRIPLETS TROP FACILES
Image 1-Anchor: min=0.000, max=238.000, mean=105.106
Normalisée: min=0.000, max=1.000
Image 1-Positive: min=0.000, max=224.884, mean=96.905
Normalisée: min=0.000, max=1.000
Image 1-Negative: min=0.000, max=255.000, mean=121.700
Normalisée: min=0.000, max=1.000
Image 2-Anchor: min=0.000, max=255.000, mean=143.764
Normalisée: min=0.000, max=1.000
Image 2-Positive: min=-0.029, max=226.629, mean=132.219
Normalisée: min=0.000, max=1.000
```

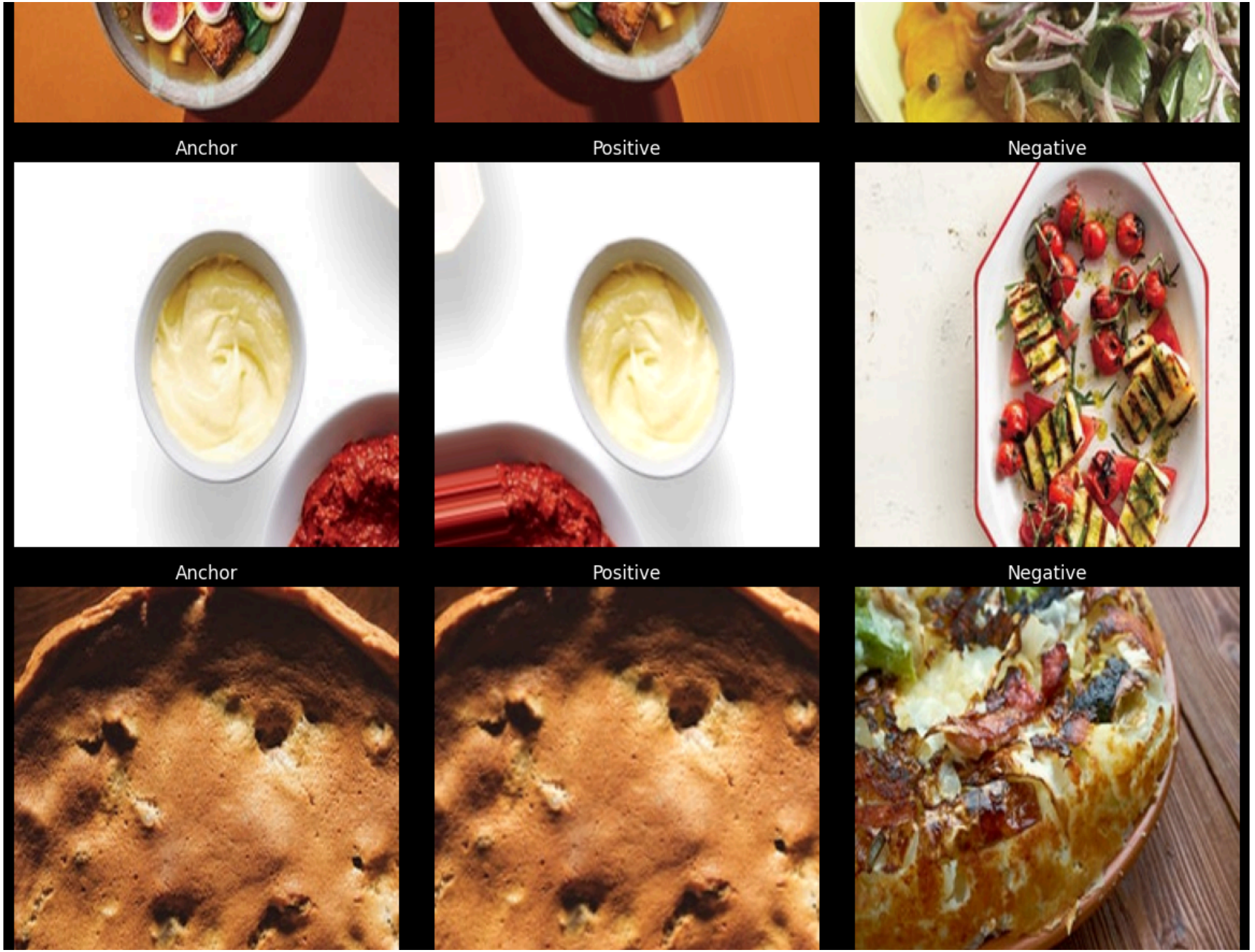
```
Image 2-Negative: min=0.000, max=255.000, mean=139.201
Normalisée: min=0.000, max=1.000
Image 3-Anchor: min=0.000, max=255.000, mean=178.561
Normalisée: min=0.000, max=1.000
Image 3-Positive: min=0.000, max=254.000, mean=187.907
Normalisée: min=0.000, max=1.000
Image 3-Negative: min=0.000, max=255.000, mean=207.141
Normalisée: min=0.000, max=1.000
Image 4-Anchor: min=0.000, max=255.000, mean=63.529
Normalisée: min=0.000, max=1.000
Image 4-Positive: min=0.000, max=265.554, mean=59.977
Normalisée: min=0.000, max=1.000
Image 4-Negative: min=22.000, max=255.000, mean=184.761
Normalisée: min=0.000, max=1.000
Image 5-Anchor: min=0.000, max=255.000, mean=221.454
Normalisée: min=0.000, max=1.000
Image 5-Positive: min=-0.045, max=240.076, mean=211.287
Normalisée: min=0.000, max=1.000
Image 5-Negative: min=0.000, max=255.000, mean=203.798
Normalisée: min=0.000, max=1.000
Image 6-Anchor: min=0.000, max=255.000, mean=188.028
Normalisée: min=0.000, max=1.000
Image 6-Positive: min=0.000, max=255.000, mean=184.165
Normalisée: min=0.000, max=1.000
Image 6-Negative: min=0.000, max=255.000, mean=120.417
Normalisée: min=0.000, max=1.000
Image 7-Anchor: min=0.000, max=255.000, mean=114.240
Normalisée: min=0.000, max=1.000
Image 7-Positive: min=0.000, max=239.000, mean=100.679
Normalisée: min=0.000, max=1.000
Image 7-Negative: min=0.000, max=255.000, mean=171.444
Normalisée: min=0.000, max=1.000
Image 8-Anchor: min=0.000, max=255.000, mean=218.373
Normalisée: min=0.000, max=1.000
Image 8-Positive: min=0.000, max=255.000, mean=216.738
Normalisée: min=0.000, max=1.000
Image 8-Negative: min=0.000, max=255.000, mean=180.737
Normalisée: min=0.000, max=1.000
Image 9-Anchor: min=0.000, max=255.000, mean=100.417
Normalisée: min=0.000, max=1.000
Image 9-Positive: min=0.000, max=255.000, mean=109.494
```

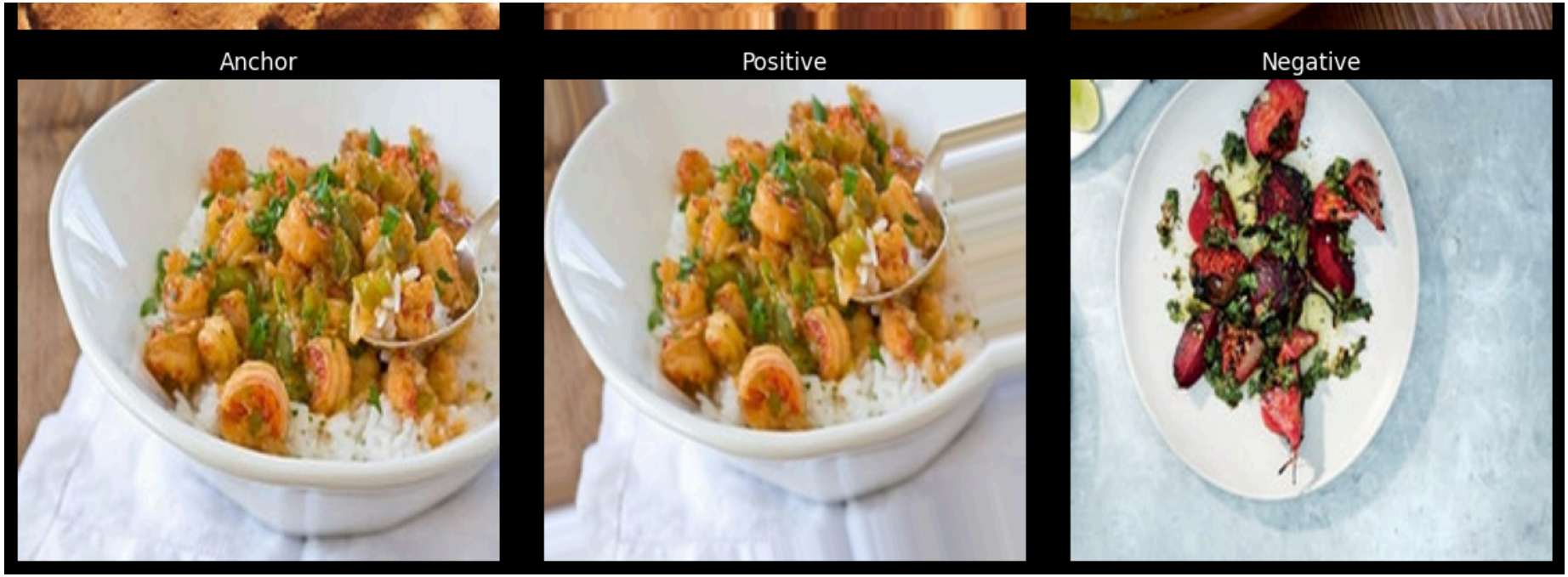
```
Normalisée: min=0.000, max=1.000
Image 9-Negative: min=0.000, max=255.000, mean=105.936
Normalisée: min=0.000, max=1.000
Image 10-Anchor: min=0.000, max=255.000, mean=168.287
Normalisée: min=0.000, max=1.000
Image 10-Positive: min=0.000, max=246.000, mean=164.757
Normalisée: min=0.000, max=1.000
Image 10-Negative: min=0.000, max=255.000, mean=186.219
Normalisée: min=0.000, max=1.000
```


Anchor
Positive
Negative
Triplets train
Anchor
Positive
Negative
Anchor
Positive
Negative


Anchor
Positive
Negative
Anchor
Positive
Negative


Anchor
Positive
Negative
Anchor
Positive
Negative


Anchor
Positive
Negative
Anchor
Positive
Negative

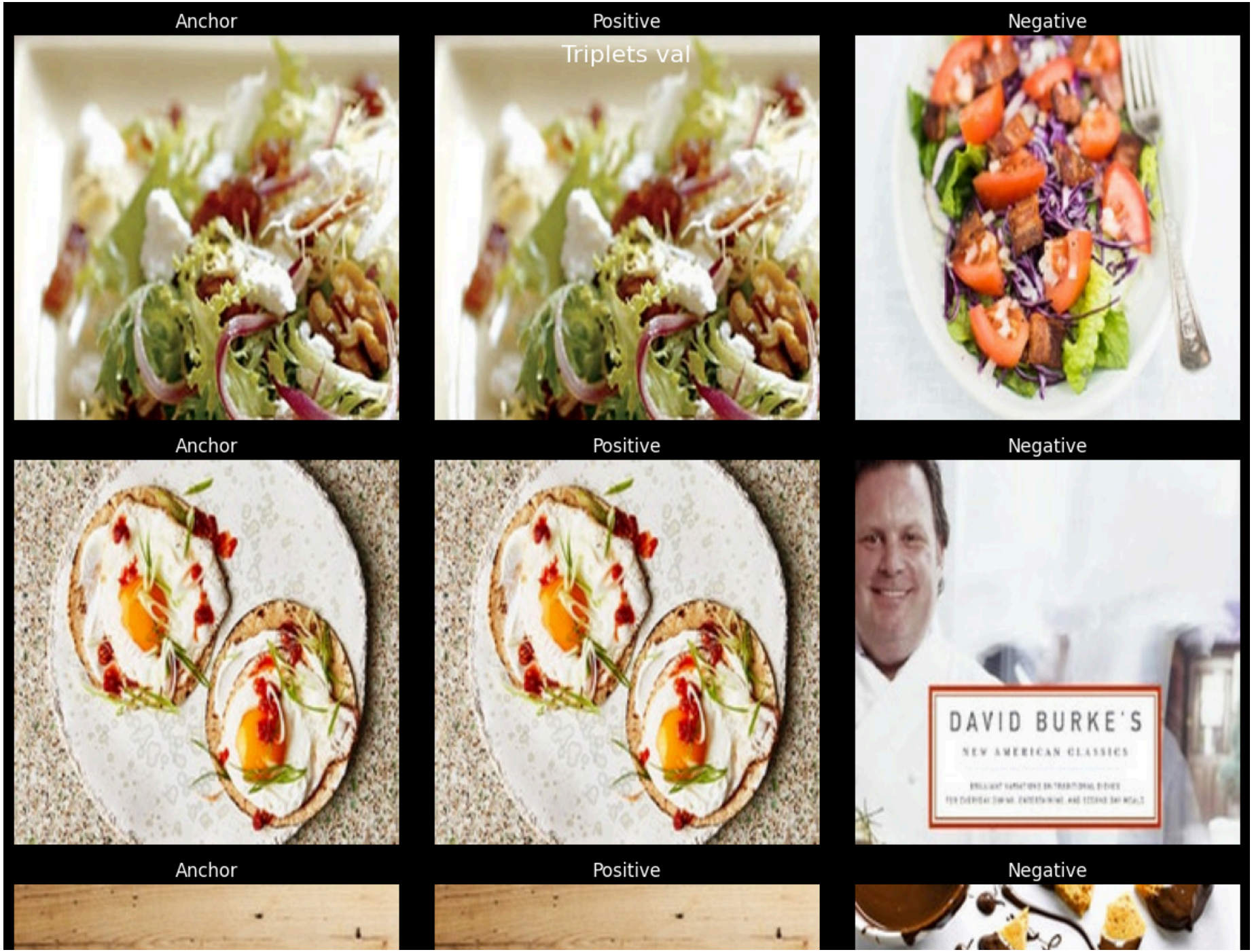


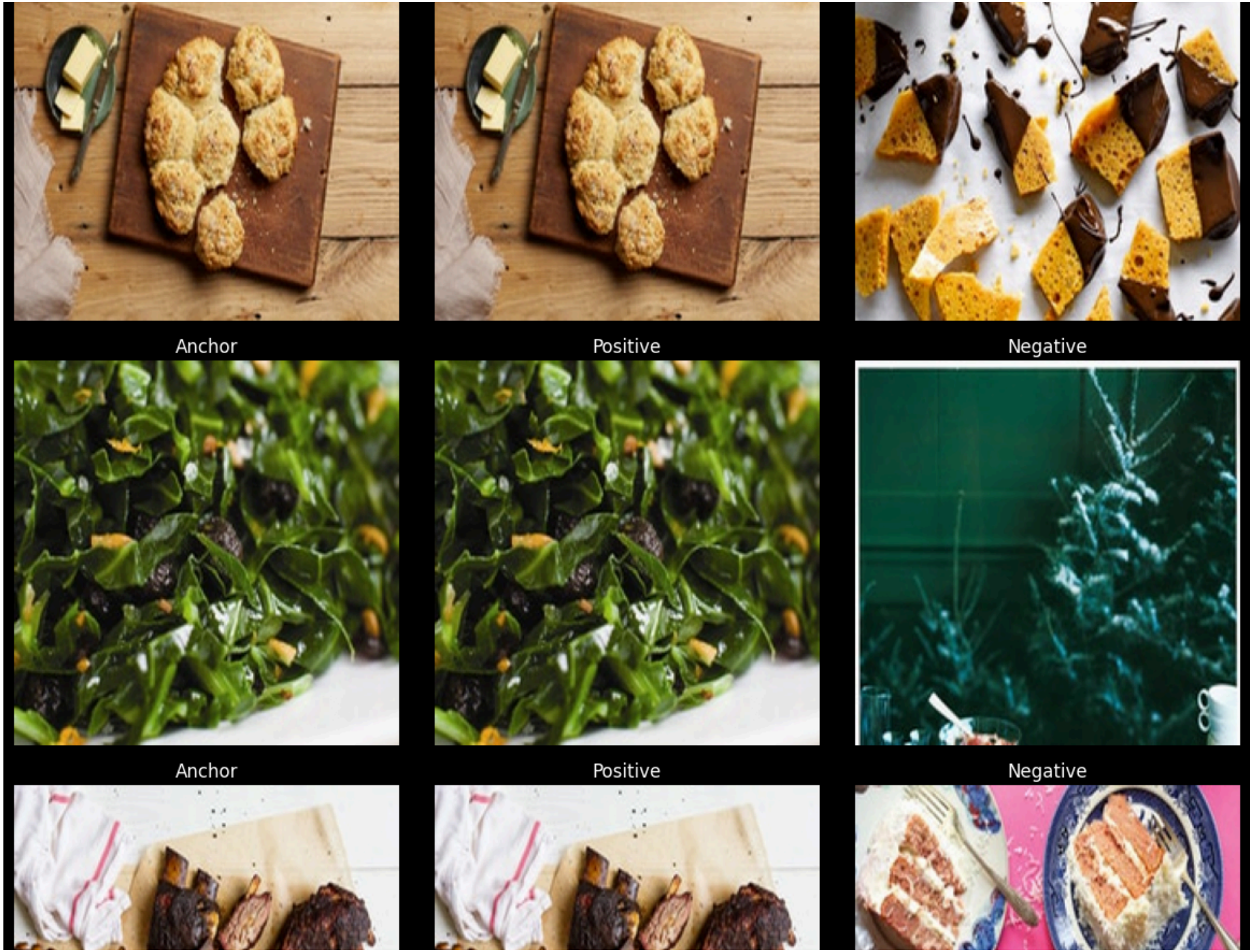

```
Filtrage des images valides...
Validation images: 100%|██████████| 13463/13463 [00:03<00:00, 3419.41it/s]
```

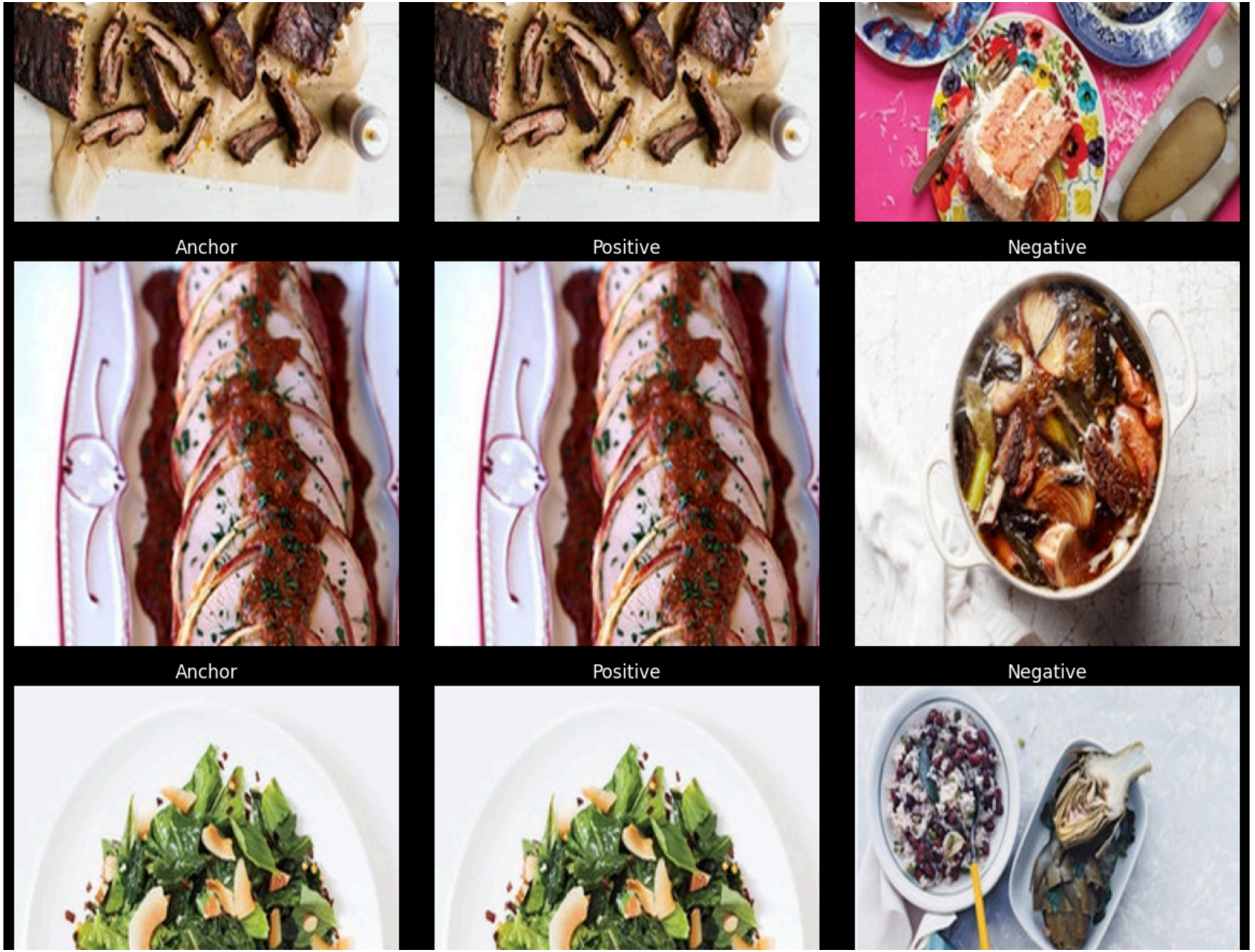
```
13463/13463 images valides
Générateur Validation créé:
- Recettes: 2655
- Images: 2689
- Augmentation: Désactivée
Format du batch triplet: (32, 3, 224, 224, 3)

=== ANALYSE DE LA DIFFICULTÉ DES TRIPLETS ===
Triplet 1:
- Distance anchor-positive: 0.0000
- Distance anchor-negative: 8040.8438
- Ratio difficulté: 804084375000.00 (FACILE)
Triplet 2:
- Distance anchor-positive: 0.0000
- Distance anchor-negative: 6904.7847
- Ratio difficulté: 690478466796.88 (FACILE)
Triplet 3:
- Distance anchor-positive: 0.0000
- Distance anchor-negative: 9834.1992
- Ratio difficulté: 983419921875.00 (FACILE)
Triplet 4:
- Distance anchor-positive: 0.0000
- Distance anchor-negative: 7225.6743
- Ratio difficulté: 722567431640.62 (FACILE)
Triplet 5:
- Distance anchor-positive: 0.0000
- Distance anchor-negative: 9349.0645
- Ratio difficulté: 934906445312.50 (FACILE)

Difficulté moyenne: 827091328125.00
Évaluation: TRIPLETS TROP FACILES
Image 1-Anchor: min=0.000, max=255.000, mean=163.989
Normalisée: min=0.000, max=1.000
Image 1-Positive: min=0.000, max=255.000, mean=163.989
Normalisée: min=0.000, max=1.000
Image 1-Negative: min=0.000, max=255.000, mean=187.575
Normalisée: min=0.000, max=1.000
Image 2-Anchor: min=0.000, max=255.000, mean=182.518
Normalisée: min=0.000, max=1.000
Image 2-Positive: min=0.000, max=255.000, mean=182.518
Normalisée: min=0.000, max=1.000
```

```
Image 2-Negative: min=9.000, max=255.000, mean=213.218
Normalisée: min=0.000, max=1.000
Image 3-Anchor: min=0.000, max=255.000, mean=136.102
Normalisée: min=0.000, max=1.000
Image 3-Positive: min=0.000, max=255.000, mean=136.102
Normalisée: min=0.000, max=1.000
Image 3-Negative: min=0.000, max=255.000, mean=140.795
Normalisée: min=0.000, max=1.000
Image 4-Anchor: min=0.000, max=255.000, mean=75.082
Normalisée: min=0.000, max=1.000
Image 4-Positive: min=0.000, max=255.000, mean=75.082
Normalisée: min=0.000, max=1.000
Image 4-Negative: min=0.000, max=255.000, mean=54.899
Normalisée: min=0.000, max=1.000
Image 5-Anchor: min=0.000, max=255.000, mean=164.474
Normalisée: min=0.000, max=1.000
Image 5-Positive: min=0.000, max=255.000, mean=164.474
Normalisée: min=0.000, max=1.000
Image 5-Negative: min=0.000, max=255.000, mean=148.704
Normalisée: min=0.000, max=1.000
Image 6-Anchor: min=0.000, max=255.000, mean=152.106
Normalisée: min=0.000, max=1.000
Image 6-Positive: min=0.000, max=255.000, mean=152.106
Normalisée: min=0.000, max=1.000
Image 6-Negative: min=0.000, max=255.000, mean=171.077
Normalisée: min=0.000, max=1.000
Image 7-Anchor: min=0.000, max=255.000, mean=178.421
Normalisée: min=0.000, max=1.000
Image 7-Positive: min=0.000, max=255.000, mean=178.421
Normalisée: min=0.000, max=1.000
Image 7-Negative: min=0.000, max=255.000, mean=178.388
Normalisée: min=0.000, max=1.000
Image 8-Anchor: min=0.000, max=255.000, mean=161.292
Normalisée: min=0.000, max=1.000
Image 8-Positive: min=0.000, max=255.000, mean=161.292
Normalisée: min=0.000, max=1.000
Image 8-Negative: min=0.000, max=255.000, mean=181.675
Normalisée: min=0.000, max=1.000
Image 9-Anchor: min=5.000, max=255.000, mean=193.666
Normalisée: min=0.000, max=1.000
Image 9-Positive: min=5.000, max=255.000, mean=193.666
```

```
Normalisée: min=0.000, max=1.000
Image 9-Negative: min=0.000, max=255.000, mean=134.611
Normalisée: min=0.000, max=1.000
Image 10-Anchor: min=0.000, max=255.000, mean=152.629
Normalisée: min=0.000, max=1.000
Image 10-Positive: min=0.000, max=255.000, mean=152.629
Normalisée: min=0.000, max=1.000
Image 10-Negative: min=0.000, max=255.000, mean=180.905
Normalisée: min=0.000, max=1.000
```


Anchor
Positive
Triplets val
Negative
Anchor
Positive
Negative
DAVID BURKE'S
NEW AMERICAN CLASSICS
Anchor
Positive
Negative


Anchor
Positive
Negative
Anchor
Positive
Negative


Anchor
Positive
Negative
Anchor
Positive
Negative


Anchor
Positive
Negative
Anchor
Positive
Negative

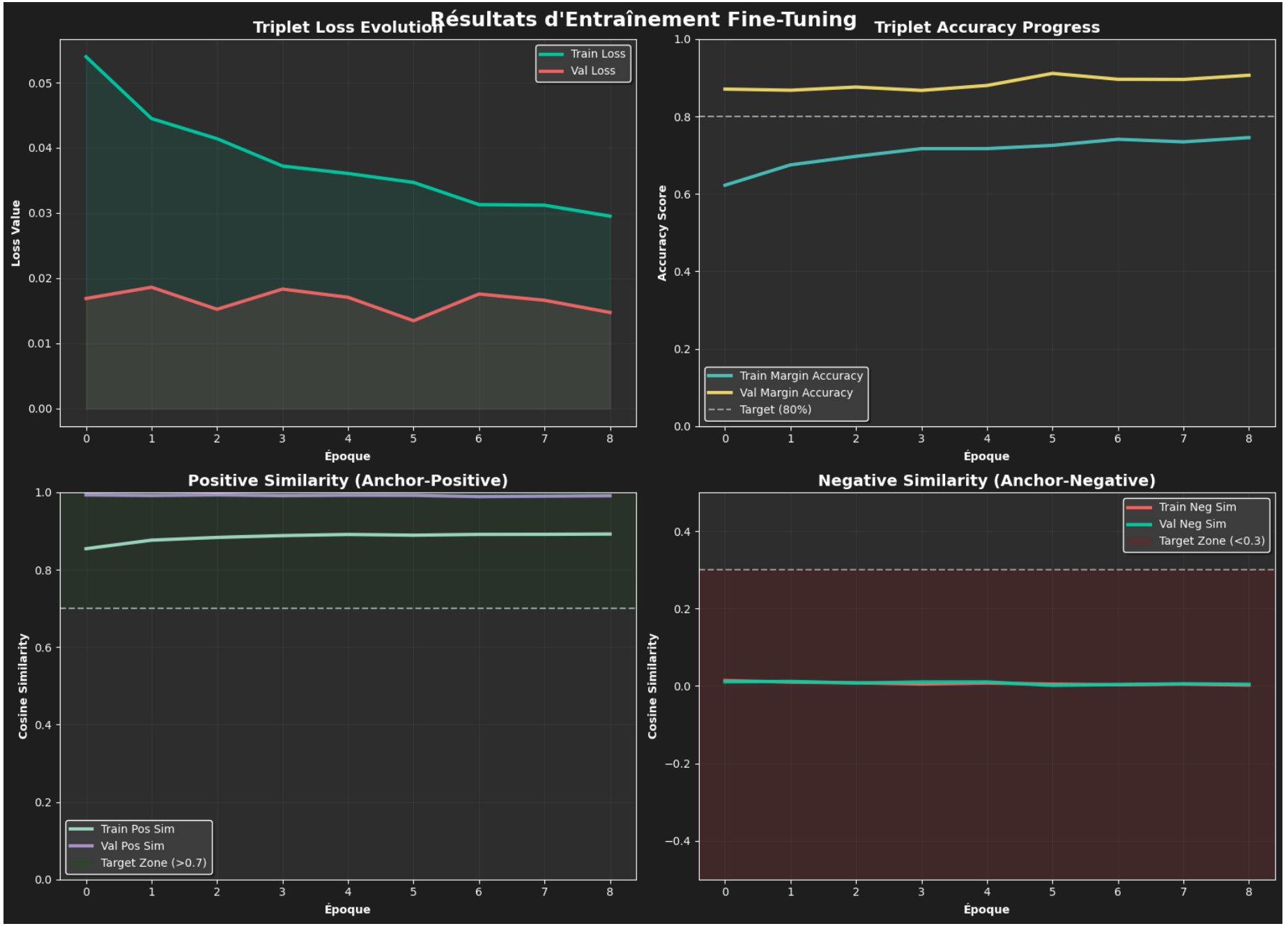

```
Modèle triplet créé:
- Input shape: (None, 3, 224, 224, 3)
- Output shape: (None, 3, 512)
Modèle compilé:
- Optimizer: Adam (lr=0.001, wd=0.0001)
- Loss: Triplet Loss (margin=0.8)
- Métriques: Accuracy + Similarités moyenne
Callbacks configurés:
- Adaptive Margin: initial=0.8
- Early Stopping: patience=3 (val_loss)
- Reduce LR: patience=2, factor=0.5
- Model Checkpoint: best_recipe_image_retrieval_model_tl.keras

============================================================
DÉBUT DE L'ENTRAÎNEMENT
============================================================
- Entraînement: 336 batch/époque
- Validation: 84 batch/époque
- Objectif: apprendre des embeddings discriminants via triplet loss
```

```
2025-07-14 16:45:41.864936: I external/local_tsl/tsl/profiler/lib/profiler_session.cc:103] Profiler session initializing.
2025-07-14 16:45:41.864965: I external/local_tsl/tsl/profiler/lib/profiler_session.cc:118] Profiler session started.
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1752511541.865868   25288 cupti_tracer.cc:1026] Profiler found 1 GPUs
W0000 00:00:1752511541.922598   25288 cupti_tracer.cc:1213] Fail to use per-thread activity buffer, cupti trace overhead may be
big. CUPTI ERROR CODE:1
2025-07-14 16:45:41.923656: I external/local_tsl/tsl/profiler/lib/profiler_session.cc:130] Profiler session tear down.
I0000 00:00:1752511541.923835   25288 cupti_tracer.cc:1249] CUPTI activity buffer flushed
Epoch 1/15
I0000 00:00:1752511574.683897   25502 service.cc:152] XLA service 0x78d220002ea0 initialized for platform CUDA (this does not g
uarantee that XLA will be used). Devices:
I0000 00:00:1752511574.683966   25502 service.cc:160]   StreamExecutor device (0): NVIDIA L4, Compute Capability 8.9
2025-07-14 16:46:16.064496: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer,
set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
I0000 00:00:1752511581.701443   25502 cuda_dnn.cc:529] Loaded cuDNN version 90300
  1/336 ━━━━━━━━━━━━━━━━━━━━ 5:16:28 57s/step - average_negative_similarity: 0.1847 - average_positive_similarity: 0.8350 - los
s: 0.1619 - triplet_margin_accuracy: 0.1562
I0000 00:00:1752511601.585660   25502 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for
the lifetime of the process.
2025-07-14 16:46:41.743606: I external/local_tsl/tsl/profiler/lib/profiler_session.cc:103] Profiler session initializing.
2025-07-14 16:46:41.743629: I external/local_tsl/tsl/profiler/lib/profiler_session.cc:118] Profiler session started.
W0000 00:00:1752511601.753907   25288 cupti_tracer.cc:1213] Fail to use per-thread activity buffer, cupti trace overhead may be
big. CUPTI ERROR CODE:1
  2/336 ━━━━━━━━━━━━━━━━━━━━ 1:13 219ms/step - average_negative_similarity: 0.1543 - average_positive_similarity: 0.8270 - los
s: 0.1439 - triplet_margin_accuracy: 0.2422
2025-07-14 16:46:41.829405: I external/local_tsl/tsl/profiler/lib/profiler_session.cc:68] Profiler session collecting data.
I0000 00:00:1752511601.879064   25288 cupti_tracer.cc:1249] CUPTI activity buffer flushed
I0000 00:00:1752511601.925463   25288 cupti_collector.cc:793]  GpuTracer has collected 686 callback api events and 685 activity
events.
I0000 00:00:1752511601.936010   25288 cupti_collector.cc:796]  GpuTracer max callback_events: 2097152, max activity events: 209
7152
2025-07-14 16:46:41.947438: I external/local_tsl/tsl/profiler/lib/profiler_session.cc:130] Profiler session tear down.
2025-07-14 16:46:41.949811: I external/local_xla/xla/tsl/profiler/rpc/client/save_profile.cc:147] Collecting XSpace to reposito
ry: ./tl/logs/recipe_image_retrieval_model_tl.keras/train/plugins/profile/2025_07_14_16_46_41/cs-01k04vb1dr61axmv8q9bx8ndc3.xpl
ane.pb
```

```
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 922ms/step - average_negative_similarity: 0.0216 - average_positive_similarity: 0.8415 - loss:
0.0617 - triplet_margin_accuracy: 0.5762
Epoch 1: val_loss improved from inf to 0.01689, saving model to ./tl/best_triplet_recipe_image_retrieval_model_tl.keras
Réduction margin à 0.75

NOUVEAU MEILLEUR MODELE D'EMBEDDINGS sauvegardé!
Fichier: ./tl/best_embedding_recipe_image_retrieval_model_tl.keras
val_loss: 0.0169
336/336 ━━━━━━━━━━━━━━━━━━━━ 421s 1s/step - average_negative_similarity: 0.0216 - average_positive_similarity: 0.8416 - loss:
0.0616 - triplet_margin_accuracy: 0.5763 - val_average_negative_similarity: 0.0104 - val_average_positive_similarity: 0.9934 -
val_loss: 0.0169 - val_triplet_margin_accuracy: 0.8709 - learning_rate: 0.0010
Epoch 2/15
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 950ms/step - average_negative_similarity: 0.0116 - average_positive_similarity: 0.8745 - loss:
0.0456 - triplet_margin_accuracy: 0.6670
Epoch 2: val_loss did not improve from 0.01689
Réduction margin à 0.7
336/336 ━━━━━━━━━━━━━━━━━━━━ 358s 1s/step - average_negative_similarity: 0.0116 - average_positive_similarity: 0.8745 - loss:
0.0456 - triplet_margin_accuracy: 0.6670 - val_average_negative_similarity: 0.0117 - val_average_positive_similarity: 0.9919 -
val_loss: 0.0186 - val_triplet_margin_accuracy: 0.8679 - learning_rate: 0.0010
Epoch 3/15
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 937ms/step - average_negative_similarity: 0.0090 - average_positive_similarity: 0.8818 - loss:
0.0430 - triplet_margin_accuracy: 0.6883
Epoch 3: val_loss improved from 0.01689 to 0.01523, saving model to ./tl/best_triplet_recipe_image_retrieval_model_tl.keras
Réduction margin à 0.6499999999999999

NOUVEAU MEILLEUR MODELE D'EMBEDDINGS sauvegardé!
Fichier: ./tl/best_embedding_recipe_image_retrieval_model_tl.keras
val_loss: 0.0152
336/336 ━━━━━━━━━━━━━━━━━━━━ 354s 1s/step - average_negative_similarity: 0.0090 - average_positive_similarity: 0.8818 - loss:
0.0430 - triplet_margin_accuracy: 0.6884 - val_average_negative_similarity: 0.0076 - val_average_positive_similarity: 0.9935 -
val_loss: 0.0152 - val_triplet_margin_accuracy: 0.8761 - learning_rate: 0.0010
Epoch 4/15
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - average_negative_similarity: 0.0039 - average_positive_similarity: 0.8870 - loss: 0.0
383 - triplet_margin_accuracy: 0.7102
Epoch 4: val_loss did not improve from 0.01523
336/336 ━━━━━━━━━━━━━━━━━━━━ 375s 1s/step - average_negative_similarity: 0.0039 - average_positive_similarity: 0.8870 - loss:
0.0383 - triplet_margin_accuracy: 0.7102 - val_average_negative_similarity: 0.0102 - val_average_positive_similarity: 0.9916 -
val_loss: 0.0183 - val_triplet_margin_accuracy: 0.8676 - learning_rate: 0.0010
Epoch 5/15
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 929ms/step - average_negative_similarity: 0.0052 - average_positive_similarity: 0.8905 - loss:
0.0356 - triplet_margin_accuracy: 0.7161
```

```
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 5: val_loss did not improve from 0.01523
336/336 ━━━━━━━━━━━━━━━━━━━━ 351s 1s/step - average_negative_similarity: 0.0052 - average_positive_similarity: 0.8905 - loss:
0.0356 - triplet_margin_accuracy: 0.7161 - val_average_negative_similarity: 0.0103 - val_average_positive_similarity: 0.9926 -
val_loss: 0.0171 - val_triplet_margin_accuracy: 0.8802 - learning_rate: 0.0010
Epoch 6/15
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 945ms/step - average_negative_similarity: 0.0056 - average_positive_similarity: 0.8896 - loss:
0.0352 - triplet_margin_accuracy: 0.7210
Epoch 6: val_loss improved from 0.01523 to 0.01347, saving model to ./tl/best_triplet_recipe_image_retrieval_model_tl.keras

NOUVEAU MEILLEUR MODELE D'EMBEDDINGS sauvegardé!
Fichier: ./tl/best_embedding_recipe_image_retrieval_model_tl.keras
val_loss: 0.0135
336/336 ━━━━━━━━━━━━━━━━━━━━ 356s 1s/step - average_negative_similarity: 0.0056 - average_positive_similarity: 0.8896 - loss:
0.0352 - triplet_margin_accuracy: 0.7210 - val_average_negative_similarity: 0.0012 - val_average_positive_similarity: 0.9924 -
val_loss: 0.0135 - val_triplet_margin_accuracy: 0.9115 - learning_rate: 5.0000e-04
Epoch 7/15
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 998ms/step - average_negative_similarity: 0.0040 - average_positive_similarity: 0.8930 - loss:
0.0315 - triplet_margin_accuracy: 0.7354
Epoch 7: val_loss did not improve from 0.01347
336/336 ━━━━━━━━━━━━━━━━━━━━ 374s 1s/step - average_negative_similarity: 0.0040 - average_positive_similarity: 0.8930 - loss:
0.0315 - triplet_margin_accuracy: 0.7354 - val_average_negative_similarity: 0.0037 - val_average_positive_similarity: 0.9891 -
val_loss: 0.0176 - val_triplet_margin_accuracy: 0.8962 - learning_rate: 5.0000e-04
Epoch 8/15
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 928ms/step - average_negative_similarity: 0.0044 - average_positive_similarity: 0.8928 - loss:
0.0303 - triplet_margin_accuracy: 0.7347
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

Epoch 8: val_loss did not improve from 0.01347
336/336 ━━━━━━━━━━━━━━━━━━━━ 350s 1s/step - average_negative_similarity: 0.0044 - average_positive_similarity: 0.8928 - loss:
0.0303 - triplet_margin_accuracy: 0.7347 - val_average_negative_similarity: 0.0056 - val_average_positive_similarity: 0.9902 -
val_loss: 0.0166 - val_triplet_margin_accuracy: 0.8958 - learning_rate: 5.0000e-04
Epoch 9/15
336/336 ━━━━━━━━━━━━━━━━━━━━ 0s 923ms/step - average_negative_similarity: 0.0019 - average_positive_similarity: 0.8923 - loss:
0.0293 - triplet_margin_accuracy: 0.7434
Epoch 9: val_loss did not improve from 0.01347
336/336 ━━━━━━━━━━━━━━━━━━━━ 347s 1s/step - average_negative_similarity: 0.0019 - average_positive_similarity: 0.8923 - loss:
0.0293 - triplet_margin_accuracy: 0.7435 - val_average_negative_similarity: 0.0042 - val_average_positive_similarity: 0.9913 -
val_loss: 0.0147 - val_triplet_margin_accuracy: 0.9066 - learning_rate: 2.5000e-04
Epoch 9: early stopping
```

```
Restoring model weights from the end of the best epoch: 6.

Entraînement terminé. Meilleur val_loss: 0.0135
Meilleur embedding model disponible: ./tl/best_embedding_recipe_image_retrieval_model_tl.keras

============================================================
 ENTRAÎNEMENT TERMINÉ AVEC SUCCÈS!
============================================================
⏱️ Temps total: 54.8 minutes
- Meilleure val_loss: 0.0135
- Précision finale: 0.9066
- Similarité pos finale: 0.9913
- Similarité neg finale: 0.0042
- Modèle sauvegardé: recipe_image_retrieval_model_tl.keras
```


Résultats d'Entraînement Fine-Tuning
Triplet Loss Evolution
Train Loss
Val Loss
Loss Value
Époque
Triplet Accuracy Progress
Accuracy Score
Train Margin Accuracy
Val Margin Accuracy
Target (80%)
Époque
Positive Similarity (Anchor-Positive)
Cosine Similarity
Train Pos Sim
Val Pos Sim
Target Zone (>0.7)
Époque
Negative Similarity (Anchor-Negative)
Cosine Similarity
Train Neg Sim
Val Neg Sim
Target Zone (<0.3)
Époque

```
RÉSUMÉ DES MÉTRIQUES FINALES:
--------------------------------------------------
Loss (validation)         : 0.0147
Margin Accuracy (validation) : 0.9066
Similarité Positive       : 0.9913
Similarité Négative       : 0.0042
Écart Pos-Neg             : 0.9871

INTERPRÉTATION:
✅ Excellente accuracy (90.7%)

SUCCÈS! Entraînement terminé.
Modèle prêt: recipe_image_retrieval_model_tl.keras
Prêt pour l'extraction d'embeddings et la recherche par similarité!
```

# 4. Création du système de recherche avec la classe RecipeImageRetrievalTL

In [13]:
```python
# PREPROCESSING
def preprocess_image(image_path, img_size=224):
    """Preprocessing robuste avec gestion d'erreurs complète + EfficientNet optimisé"""
    try:
        if not os.path.exists(image_path):
            return None

        # Vérifier la taille du fichier (éviter fichiers corrompus)
        if os.path.getsize(image_path) < 1024:  # Moins de 1KB
            return None

        img = Image.open(image_path)

        # Vérifier si l'image est valide
        img.verify()
        img = Image.open(image_path)  # Recharger après verify

        # Convertir en RGB si nécessaire
        if img.mode != 'RGB':
            img = img.convert('RGB')

        # Redimensionner avec qualité optimale
```

```
        img = img.resize((img_size, img_size), Image.Resampling.LANCZOS)

        # Convertir en array et appliquer preprocessing EfficientNet
        img_array = np.array(img, dtype=np.float32)
        img_array = preprocess_input(img_array)  #  Normalisation EfficientNet native

        return img_array

    except Exception as e:
        return None

#  COUCHE L2 NORMALIZATION PERSONNALISÉE (SÉRIALISABLE)
class L2NormalizationLayer(Layer):
    """Couche personnalisée pour normalisation L2 - sérialisable"""

    def __init__(self, axis=1, **kwargs):
        super(L2NormalizationLayer, self).__init__(**kwargs)
        self.axis = axis

    def call(self, inputs):
        return tf.nn.l2_normalize(inputs, axis=self.axis)

    def compute_output_shape(self, input_shape):
        return input_shape

    def get_config(self):
        config = super(L2NormalizationLayer, self).get_config()
        config.update({'axis': self.axis})
        return config

#  COUCHE EXTRACTION TRIPLET PERSONNALISÉE (SÉRIALISABLE)
class ExtractTripletComponent(Layer):
    """Couche pour extraire une composante du triplet"""

    def __init__(self, component_index, **kwargs):
        super(ExtractTripletComponent, self).__init__(**kwargs)
        self.component_index = component_index

    def call(self, inputs):
        return inputs[:, self.component_index]
```

```
    def compute_output_shape(self, input_shape):
        # input_shape = (batch_size, 3, height, width, channels)
        return (input_shape[0], input_shape[2], input_shape[3], input_shape[4])

    def get_config(self):
        config = super(ExtractTripletComponent, self).get_config()
        config.update({'component_index': self.component_index})
        return config

#  COUCHE STACK TRIPLET PERSONNALISÉE (SÉRIALISABLE)
class TripletStackLayer(Layer):
    """Couche personnalisée pour empiler les embeddings triplet"""

    def __init__(self, **kwargs):
        super(TripletStackLayer, self).__init__(**kwargs)

    def call(self, inputs):
        # inputs = [anchor_emb, positive_emb, negative_emb]
        return tf.stack(inputs, axis=1)

    def compute_output_shape(self, input_shape):
        # input_shape = [(batch_size, embedding_dim), ...]
        batch_size = input_shape[0][0]
        embedding_dim = input_shape[0][1]
        return (batch_size, 3, embedding_dim)

    def get_config(self):
        return super(TripletStackLayer, self).get_config()

class RecipeImageRetrievalTL:
    """
    Version Transfer Learning de RecipeImageRetrievalTL
    Même fonctionnalités, mais avec le modèle TL entraîné
    """

    def __init__(self, model_path=None, recipes_df=None):
        #  Charger le modèle TL
        if model_path is None:
            model_path = './tl/best_embedding_recipe_image_retrieval_model_tl.keras'

        #  TRIPLET LOSS OPTIMISÉE POUR EMBEDDINGS NORMALISÉS
```

```
        def triplet_loss(margin=0.3):
            """Triplet loss optimisée pour embeddings L2-normalisés"""
            def triplet_loss_fn(y_true, y_pred):
                """
                y_pred contient [anchor, positive, negative] embeddings normalisés
                Utilise distance cosinus (1 - similarity) pour la loss
                """
                anchor = y_pred[:, 0, :]      # (batch_size, embedding_dim)
                positive = y_pred[:, 1, :]    # (batch_size, embedding_dim)
                negative = y_pred[:, 2, :]    # (batch_size, embedding_dim)

                # Calcul des distances cosinus (1 - similarité cosinus)
                # Pour embeddings normalisés: cosine_sim = dot_product
                pos_similarity = tf.reduce_sum(anchor * positive, axis=1)
                neg_similarity = tf.reduce_sum(anchor * negative, axis=1)

                pos_distance = 1.0 - pos_similarity
                neg_distance = 1.0 - neg_similarity

                # Triplet loss: max(0, pos_dist - neg_dist + margin)
                basic_loss = pos_distance - neg_distance + margin
                loss = tf.maximum(0.0, basic_loss)

                return tf.reduce_mean(loss)

            return triplet_loss_fn

        #  MÉTRIQUES
        def triplet_margin_accuracy(y_true, y_pred):
            """% de triplets qui respectent la marge complète (vraie réussite)"""
            anchor = y_pred[:, 0, :]
            positive = y_pred[:, 1, :]
            negative = y_pred[:, 2, :]

            pos_similarity = tf.reduce_sum(anchor * positive, axis=1)
            neg_similarity = tf.reduce_sum(anchor * negative, axis=1)

            # Distance cosinus
            pos_distance = 1.0 - pos_similarity
            neg_distance = 1.0 - neg_similarity
```

```
        # Triplet satisfait si : neg_dist - pos_dist > margin
        margin = CONFIG_TL['TRIPLET_MARGIN']
        margin_satisfied = tf.cast((neg_distance - pos_distance) > margin, tf.float32)

        return tf.reduce_mean(margin_satisfied)

    def average_positive_similarity(y_true, y_pred):
        """Similarité moyenne anchor-positive"""
        anchor = y_pred[:, 0, :]
        positive = y_pred[:, 1, :]
        pos_similarity = tf.reduce_sum(anchor * positive, axis=1)
        return tf.reduce_mean(pos_similarity)

    def average_negative_similarity(y_true, y_pred):
        """Similarité moyenne anchor-negative"""
        anchor = y_pred[:, 0, :]
        negative = y_pred[:, 2, :]
        neg_similarity = tf.reduce_sum(anchor * negative, axis=1)
        return tf.reduce_mean(neg_similarity)

    # Custom objects pour charger correctement le modèle
    custom_objects = {
        'L2NormalizationLayer': L2NormalizationLayer,
        'ExtractTripletComponent': ExtractTripletComponent,
        'TripletStackLayer': TripletStackLayer,
        'triplet_loss': triplet_loss(),
        'triplet_margin_accuracy': triplet_margin_accuracy,
        'average_positive_similarity': average_positive_similarity,
        'average_negative_similarity': average_negative_similarity
    }

    self.model = tf.keras.models.load_model(model_path,
                                            compile=False,
                                            safe_mode=False,
                                            custom_objects=custom_objects)

    print(f"Modèle TL chargé: {model_path}")

    # DataFrame
    if recipes_df is None:
        recipes_df = pd.read_pickle("./data/recipes_with_images_dataframe.pkl")
```

```
        self.recipes_df = recipes_df
        self.embeddings_db = None
        self.image_paths = None
        self.image_to_recipe_map = {}

        #  Config
        self.config = CONFIG_TL

    def prepare_database(self):
        """IDENTIQUE au notebook raw"""
        print("Préparation de la base de données d'images...")

        valid_recipes = self.recipes_df[self.recipes_df['image_path'].notna()].copy()
        self.image_paths = valid_recipes['image_path'].tolist()

        for idx, row in valid_recipes.iterrows():
            self.image_to_recipe_map[row['image_path']] = {
                'title': row['Title'],
                'ingredients': row['Ingredients'],
                'instructions': row['Instructions'],
                'original_index': idx
            }

        print("Image-to-recipe mapping créé!")

    def build_embeddings_database(self, batch_size=32):
        """IDENTIQUE - juste utilise le modèle TL"""
        if self.image_paths is None:
            self.prepare_database()

        print("Construction de la base de données d'embeddings...")

        embeddings = []
        total_batches = len(self.image_paths) // batch_size + (1 if len(self.image_paths) % batch_size else 0)

        for i in range(0, len(self.image_paths), batch_size):
            batch_paths = self.image_paths[i:i+batch_size]
            batch_images = []

            print(f"BATCH {i//batch_size + 1}/{total_batches}...")
```

```
        for img_path in batch_paths:
            try:
                # Utilise le même preprocessing que TL
                img_array = preprocess_image(img_path, self.config['IMG_SIZE'])
                if img_array is not None:
                    batch_images.append(img_array)
                else:
                    batch_images.append(np.zeros((self.config['IMG_SIZE'], self.config['IMG_SIZE'], 3)))
            except:
                batch_images.append(np.zeros((self.config['IMG_SIZE'], self.config['IMG_SIZE'], 3)))

        if batch_images:
            batch_images = np.array(batch_images)
            batch_embeddings = self.model.predict(batch_images, verbose=0)

            ### DEBUT DEBUG POUR VOIR SI LES NORM L2 SONT OK AU CHARGEMENT DU MODELE TL ###
            # Norme d'un embedding individuel
            print(f"Norme 1er embedding: {np.linalg.norm(batch_embeddings[0])}")
            print(f"Normes de tous: {np.linalg.norm(batch_embeddings, axis=1)[:5]}")

            # Vérification moyenne
            mean_norm = np.mean(np.linalg.norm(batch_embeddings, axis=1))
            print(f"Norme moyenne: {mean_norm}")
            ### FIN DEBUG ###

            embeddings.extend(batch_embeddings)

    self.embeddings_db = np.array(embeddings)
    print(f"Transfer Learning embeddings database built: {len(self.embeddings_db)} embeddings")
    print(f"Database shape: {self.embeddings_db.shape}")


def visualize_model_architecture(self):
    """Visualisation de l'architecture du modèle Transfer Learning"""
    fig = plt.figure(figsize=(16, 12))
    fig.patch.set_facecolor('#1a1a1a')

    fig.suptitle('Architecture du modèle Transfer Learning',
                 fontsize=18, color='white', fontweight='bold', y=0.94)
```

```
        # Get model parameters info
        total_params = self.model.count_params()
        trainable_params = sum(layer.count_params() for layer in self.model.layers if layer.trainable)
        frozen_params = total_params - trainable_params

        total_recipes = len(self.recipes_df)
        recipes_with_images = len(self.recipes_df[self.recipes_df['image_path'].notna()]) if 'image_path' in self.recipes_df.c
        embeddings_ready = len(self.embeddings_db) if self.embeddings_db is not None else 0

        # Distribution des paramètres
        ax1 = plt.subplot(2, 2, 1)
        ax1.set_facecolor('#2d2d2d')

        if trainable_params > 0:
            params_data = [frozen_params, trainable_params]
            params_labels = ['Gelés (EfficientNet)', 'Entraînables (Custom Head)']
            colors_params = ['#4ECDC4', '#FF6B9D']
        else:
            params_data = [total_params]
            params_labels = ['Tous gelés']
            colors_params = ['#4ECDC4']

        wedges, texts, autotexts = ax1.pie(params_data, labels=params_labels, colors=colors_params,
                                           autopct='%1.1f%%', startangle=90)

        for text in texts:
            text.set_color('white')
        for autotext in autotexts:
            autotext.set_color('white')
            autotext.set_fontweight('bold')

        ax1.set_title(f'Distribution des paramètres\nTotal: {total_params:,}',
                      fontsize=14, color='white', fontweight='bold', pad=15)

        # Pipeline de Transfer Learning
        ax2 = plt.subplot(2, 2, 2)
        ax2.set_facecolor('#2d2d2d')
        ax2.set_xlim(0, 12)
        ax2.set_ylim(0, 8)

        pipeline_steps = [
```

```
        (1.5, 5, "Image\n224x224x3", '#FF6B9D', 1.5),
        (3.8, 5, "EfficientNet\n(Gelés)", '#4ECDC4', 2.0),
        (6.8, 5, "Custom Head\n(Entraînables)", '#FF9F43', 2.0),
        (9.8, 5, "Embedding\n512-dim", '#FECA57', 1.6)
    ]

    # Dessiner les étapes avec cases plus grandes
    for i, (x, y, text, color, width) in enumerate(pipeline_steps):
        ax2.add_patch(plt.Rectangle((x-width/2, y-0.9), width, 1.8,
                                    facecolor=color, alpha=0.8, edgecolor='white', linewidth=2))
        ax2.text(x, y, text, ha='center', va='center', fontsize=10,
                color='white', fontweight='bold')

        # Flèches entre les étapes
        if i < len(pipeline_steps) - 1:
            next_x = pipeline_steps[i+1][0]
            next_width = pipeline_steps[i+1][4]
            ax2.arrow(x + width/2 + 0.1, y, next_x - x - width/2 - next_width/2 - 0.2, 0,
                    head_width=0.3, head_length=0.2, fc='white', ec='white', alpha=0.8)

    # Workflow explicatif - Case plus grande
    ax2.add_patch(plt.Rectangle((0.5, 2.3), 10.5, 1.8,
                                facecolor='#1a1a1a', alpha=0.9, edgecolor='#45B7D1', linewidth=2))

    workflow_text = """
    ENTRAINEMENT: 3x ce pipeline (anchor, positive, negative) → Triplet Loss
    INFERENCE: 1x ce pipeline → Recherche dans la base de données d'embeddings
    """

    ax2.text(5.75, 3.2, workflow_text, ha='center', va='center', fontsize=10,
            color='white', fontweight='bold')

    # Titre sans icônes
    ax2.set_title('Architecture et workflow du Transfer Learning', fontsize=14, color='white', fontweight='bold', pad=15)

    # Note simple en bas
    ax2.text(5.75, 0.8, "Même architecture, différent usage: Entraînement vs Inférence",
            ha='center', va='center', fontsize=9, color='#45B7D1', style='italic')

    ax2.axis('off')
```

```
        # Statistiques d'entrainement
        ax3 = plt.subplot(2, 2, 3)
        ax3.set_facecolor('#2d2d2d')

        training_stats = {
            'Total recettes': total_recipes,
            'Avec\nimages': recipes_with_images,
            'Embeddings TL\nprêts': embeddings_ready,
            'Custom\nDim': self.config['EMBEDDING_DIM_CUSTOM']
        }

        bars = ax3.bar(list(training_stats.keys()), list(training_stats.values()),
                       color=['#FF6B9D', '#4ECDC4', '#45B7D1', '#96CEB4'],
                       alpha=0.9, edgecolor='white', linewidth=2)

        ax3.set_title('Statistiques du Transfer Learning', fontsize=14, color='white', fontweight='bold', pad=15)
        ax3.set_ylabel('Count', fontsize=12, color='white')
        ax3.tick_params(axis='x', labelsize=10, colors='white')
        ax3.tick_params(axis='y', colors='white')
        ax3.grid(True, alpha=0.3, color='#404040')

        for bar, value in zip(bars, training_stats.values()):
            ax3.text(bar.get_x() + bar.get_width()/2., bar.get_height() + max(training_stats.values()) * 0.01,
                    f'{value:,}', ha='center', va='bottom', color='white', fontweight='bold')

        # Résumé
        ax4 = plt.subplot(2, 2, 4)
        ax4.set_facecolor('#2d2d2d')
        ax4.axis('off')

        if embeddings_ready > 0:
            status = " Prêt pour la recherche par similarité"
        elif recipes_with_images > 0:
            status = "Prêt pour extraire les embeddings"
        else:
            status = "Aucune image"

        combined_text = f"""
        RÉSUMÉ DU MODÈLE DE TRANSFER LEARNING:

        Architecture:
```

```
• Base: EfficientNetB0 (gélé avec poids ImageNet)
• Custom Head: 1280→1024→512 (entraïnée)
• Fonction de loss: Triplet Loss (margin={self.config['TRIPLET_MARGIN']})
• Régularisation: Dropout ({self.config['DROPOUT_RATE']})

Custom Layers (Serializable):
• L2NormalizationLayer: Normalisation L2
• ExtractTripletComponent: Extraction des triplets
• TripletStackLayer: Stack des embeddings pour le loss
• Toutes les couches sont compatibles avec le sauvegarde et le chargement du modèle

Paramètres:
• Total: {total_params:,}
• Gelés: {frozen_params:,} ({frozen_params/total_params*100:.1f}%)
• Entraînables: {trainable_params:,} ({trainable_params/total_params*100:.1f}%)

Configuration de l'entrainement:
• Epoques: {self.config['TRANSFER_EPOCHS']} (early stopping à l'époque 9)
• Learning Rate: {self.config['TRANSFER_LR']} (optimisé pour TL)
• Taille des batch: {self.config['BATCH_SIZE']} (bon compromis mémoire/performance)
• Triplet Generator avec hard negative sampling + triplet_margin_accuracy
• Output: {self.config['EMBEDDING_DIM_CUSTOM']}-dim

Callbacks d'entrainement:
• AdaptiveMarginCallback: Margin adaptatif
• EarlyStopping: Patience {self.config['PATIENCE']}
• ReduceLROnPlateau: Factor {self.config['REDUCE_LR_FACTOR']}
• EmbeddingModelCheckpoint: Custom callback
(sauvegarde modèle d'embeddings, pas du modèle de triplets)

Performances attendues:
• Apprentissage de caractéristiques spécifiques aux recettes via triplet loss
• Architecture personnalisée avec couches sérialisables
• Entraînement robuste avec callbacks avancés
• Meilleur modèle époque 6
• triplet_margin_accuracy: 90.7%
• Supérieur au modèle "raw" : compréhension de la sémantique des recettes

Statut: {status}
Dataset: {total_recipes:,} recettes, {recipes_with_images:,} avec images
"""
```

```
    ax4.text(0.05, 0.95, combined_text, transform=ax4.transAxes, fontsize=9,
            color='white', verticalalignment='top', fontfamily='monospace',
            bbox=dict(boxstyle="round,pad=0.5", facecolor='#2d2d2d', alpha=0.8))

    plt.tight_layout(pad=2.0, rect=[0, 0.02, 1, 0.92])
    plt.show()

    print("\n" + "="*80)
    print("DÉTAILS DU TRANSFER LEARNING:")
    print("="*80)
    print(f"- EfficientNetB0 gelé: {frozen_params:,} paramètres")
    print(f"- Custom Head entrainable: {trainable_params:,} paramètres")
    print(f"- Total: {total_params:,} paramètres")
    print(f"- Ratio d'entraînement: {trainable_params/total_params*100:.1f}%")
    print(f"- Custom Head: 1280→1024→512 (ENTRAÎNABLES)")


def search_similar_recipes(self, query_image_path, top_k=3):
    """IDENTIQUE - utilise cosine similarity"""
    if self.embeddings_db is None:
        print("Il faut d'abord construire la base de données d'embeddings!")
        return None

    print(f"Recherche des {top_k} recette similaires avec le modèle de Transfer Learning...")

    try:
        #  Même preprocessing que TL
        img_array = preprocess_image(query_image_path, self.config['IMG_SIZE'])
        if img_array is None:
            raise Exception("Impossible de charger l'image")

        img_array = np.expand_dims(img_array, axis=0)
        query_embedding = self.model.predict(img_array, verbose=0)
    except Exception as e:
        print(f"Erreur lors du chargement de l'image: {e}")
        return None

    # Calcul similarité cosinus
    similarities = cosine_similarity(query_embedding, self.embeddings_db)[0]
```

```
        # Get top-k similaires
        top_indices = np.argsort(similarities)[::-1][:top_k]

        results = []
        for rank, idx in enumerate(top_indices):
            img_path = self.image_paths[idx]
            similarity_score = similarities[idx]

            if img_path in self.image_to_recipe_map:
                recipe_info = self.image_to_recipe_map[img_path].copy()
                recipe_info['rank'] = rank + 1
                recipe_info['similarity'] = similarity_score
                recipe_info['similar_image_path'] = img_path
                results.append(recipe_info)

        return results

    def display_results(self, query_image_path, results):
        """IDENTIQUE - même visualisation"""
        if not results:
            print("Aucun résultat")
            return

        fig, axes = plt.subplots(1, len(results) + 1, figsize=(20, 6))
        fig.patch.set_facecolor('#1a1a1a')
        fig.suptitle('IMAGE → TOP 3 RECETTES SIMILAIRES (Transfer Learning)',
                     fontsize=18, color='white', fontweight='bold', y=0.95)

        colors = ['#FF6B9D', '#4ECDC4', '#45B7D1', '#96CEB4']

        try:
            query_img = Image.open(query_image_path)
            axes[0].imshow(query_img)
            axes[0].set_title("INPUT", fontsize=14, color=colors[0], fontweight='bold', pad=15)
            axes[0].axis('off')

            for spine in axes[0].spines.values():
                spine.set_visible(True)
                spine.set_color(colors[0])
                spine.set_linewidth(3)
        except:
```

```
            axes[0].text(0.5, 0.5, " Query image\nnot found", ha='center', va='center',
                         transform=axes[0].transAxes, color='white', fontsize=12)

        for i, result in enumerate(results):
            try:
                result_img = Image.open(result['similar_image_path'])
                axes[i+1].imshow(result_img)

                title = result['title'][:25] + "..." if len(result['title']) > 25 else result['title']
                similarity = result['similarity']

                axes[i+1].set_title(f"#{result['rank']}: {title}\n TL Similarity: {similarity:.3f}",
                                    fontsize=12, color=colors[i+1], fontweight='bold', pad=15)
                axes[i+1].axis('off')

                for spine in axes[i+1].spines.values():
                    spine.set_visible(True)
                    spine.set_color(colors[i+1])
                    spine.set_linewidth(3)

            except:
                axes[i+1].text(0.5, 0.5, f"Image #{result['rank']}\nintrouvable",
                               ha='center', va='center', transform=axes[i+1].transAxes,
                               color='white', fontsize=11)

        plt.tight_layout()
        plt.show()

        print("\n" + "="*80)
        print("RECETTES SIMILAIRES (Transfer Learning)")
        print("="*80)

        for result in results:
            print(f"\nRANG #{result['rank']}: {result['similarity']:.4f}")
            print(f"{result['title']}")
            print("-" * 60)

            ingredients = str(result['ingredients'])
            if len(ingredients) > 300:
                ingredients = ingredients[:300] + "..."
            print(f"Ingredients:\n{ingredients}")
```

```
            print("-" * 60)

            instructions = str(result['instructions'])
            if len(instructions) > 400:
                instructions = instructions[:400] + "..."
            print(f"Instructions:\n{instructions}")
            print("=" * 80)

    def test_with_random_image(self, top_k=3):
        """IDENTIQUE"""
        if not self.image_paths:
            print("Aucune image disponible pour le test")
            return

        random_idx = np.random.randint(0, len(self.image_paths))
        test_image_path = self.image_paths[random_idx]

        print(f"Test avec l'image: {test_image_path}")

        results = self.search_similar_recipes(test_image_path, top_k)

        if results:
            self.display_results(test_image_path, results)
        else:
            print("Aucun résultat")

print("RecipeImageRetrievalTL créée!")
print("Utilisation:")
print("retrieval_tl = RecipeImageRetrievalTL()")
print("retrieval_tl.build_embeddings_database()")
print("retrieval_tl.test_with_random_image()")
print("retrieval_tl.search_similar_recipes(test_image_path, top_k)")
```

```
RecipeImageRetrievalTL créée!
Utilisation:
retrieval_tl = RecipeImageRetrievalTL()
retrieval_tl.build_embeddings_database()
retrieval_tl.test_with_random_image()
retrieval_tl.search_similar_recipes(test_image_path, top_k)
```

# 5. Extraction des embeddings avec le nouveau modèle !

In [10]:
```
# ÉTAPE 1: vérifier le modèle TL
model_tl_path = './tl/best_embedding_recipe_image_retrieval_model_tl.keras'
if os.path.exists(model_tl_path):
    print(f"Modèle TL trouvé: {model_tl_path}")
else:
    print(f"Modèle TL non trouvé: {model_tl_path}")
    print("Veuillez d'abord exécuter les cellules d'entraînement pour créer le modèle TL")

# ÉTAPE 2: vérifier DataFrame
data_path = "./data/recipes_with_images_dataframe.pkl"
if os.path.exists(data_path):
    try:
        recipes_df = pd.read_pickle(data_path)
        print(f"DataFrame chargée avec {len(recipes_df)} recettes")
    except Exception as e:
        print(f"Erreur dans le chargement du DataFrame: {e}")
        recipes_df = None
else:
    print(f"DataFrame introuvable: {data_path}")
    print("Veuillez d'abord exécuter le notebook 'recipe_image_retrieval_raw.ipynb' pour créer le DataFrame")
    recipes_df = None

# ÉTAPE 3: Initialiser le système TL
if os.path.exists(model_tl_path) and recipes_df is not None:
    print("\nInitialisation du système de recherche par similarité avec Transfer Learning...")

    # Chargement du modèle TL
    try:
        print("Le modèle TL a été chargé avec succès!")

        # Initialiser le système de retrieval TL
        retrieval_system_tl = RecipeImageRetrievalTL(model_tl_path, recipes_df)

        # Préparer la base de données
        retrieval_system_tl.prepare_database()
```

```
        print("Le système de recherche par similarité avec Transfer Learning a été initialisé avec succès!")
        print(f"La base de données contient {len(retrieval_system_tl.image_paths)} images")

        print(f"\n Détails du modèle TL:")
        print(f"- Input shape: {retrieval_system_tl.model.input_shape}")
        print(f"- Output shape: {retrieval_system_tl.model.output_shape}")
        print(f"- Embedding dim: {CONFIG_TL['EMBEDDING_DIM_CUSTOM']}")
        print(f"- Entraînement: Triplet Loss optimisé")

    except Exception as e:
        print(f"Erreur dans le chargement du modèle TL: {e}")
        print("Vérifiez si toutes les couches personnalisées sont correctement définies")
        retrieval_system_tl = None

else:
    print("Impossible d'initialiser le système de recherche par similarité avec Transfer Learning!")
    print("Fichiers manquants ou incorrects.")
    retrieval_system_tl = None

#  ÉTAPE 4: construire la base de données d'embeddings TL
if retrieval_system_tl is not None:
    print("\nConstruction de la base de données d'embeddings avec Transfer Learning...")
    print("Cela peut prendre quelques minutes...")

    # Batch size adapté pour TL
    retrieval_system_tl.build_embeddings_database(batch_size=CONFIG_TL['BATCH_SIZE'])

    print("\nLe système de recherche par similarité avec Transfer Learning est prêt!")
    print(f" {len(retrieval_system_tl.embeddings_db)} embeddings générés")
    print(f"Shape embeddings: {retrieval_system_tl.embeddings_db.shape}")

    tl_embeddings_path = f"./tl/{CONFIG_TL['EMBEDDINGS_NAME']}"
    tl_metadata_path = f"./tl/{CONFIG_TL['METADATA_NAME']}"

    np.save(tl_embeddings_path, retrieval_system_tl.embeddings_db)

    with open(tl_metadata_path, 'wb') as f:
        pickle.dump({
            'image_paths': retrieval_system_tl.image_paths,
            'image_to_recipe_map': retrieval_system_tl.image_to_recipe_map,
            'config': CONFIG_TL
```

```
        }, f)

    print(f"Embeddings sauvegardés: {tl_embeddings_path}")
    print(f"Metadonnées sauvegardés: {tl_metadata_path}")

    print("\nProchaines étapes:")
    print("1. Tester avec une image aléatoire: retrieval_system_tl.test_with_random_image()")
    print("2. Rechercher avec votre propre image: retrieval_system_tl.search_similar_recipes('path/to/image.jpg')")
    print("3. Comparer avec les résultats avec le modèle raw")

else:
    print("\nSystème de recherche par similarité avec Transfer Learning n'est pas initialisé.")
    print("Dépannage:")
    print("1. Vérifiez que vous avez exécuté les cellules d'entraînement TL")
    print("2. Vérifiez que les couches personnalisées sont bien définies")
    print("3. Vérifiez que le fichier du modèle existe")
    print("4. Vérifiez que le DataFrame des recettes est disponible")
```

```
Modèle TL trouvé: ./tl/best_embedding_recipe_image_retrieval_model_tl.keras
DataFrame chargée avec 13463 recettes

Initialisation du système de recherche par similarité avec Transfer Learning...
Le modèle TL a été chargé avec succès!
```

```
Modèle TL chargé: ./tl/best_embedding_recipe_image_retrieval_model_tl.keras
Préparation de la base de données d'images...
Image-to-recipe mapping créé!
Le système de recherche par similarité avec Transfer Learning a été initialisé avec succès!
La base de données contient 13463 images

 Détails du modèle TL:
- Input shape: (None, 224, 224, 3)
- Output shape: (None, 512)
- Embedding dim: 512
- Entraînement: Triplet Loss optimisé

Construction de la base de données d'embeddings avec Transfer Learning...
Cela peut prendre quelques minutes...
Construction de la base de données d'embeddings...
BATCH 1/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 2/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 3/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 4/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 5/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 6/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 7/421...
Norme 1er embedding: 1.0
```

```
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 8/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 9/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 10/421...
Norme 1er embedding: 1.0000001192092896
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 11/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 12/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.9999999  1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 13/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 14/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 15/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 16/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 17/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
```

```
Norme moyenne: 1.0
BATCH 18/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 19/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 20/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 21/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 22/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 23/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 24/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 25/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 26/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 27/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
```

```
BATCH 28/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 29/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 30/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.9999999  0.99999994]
Norme moyenne: 1.0
BATCH 31/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 32/421...
Norme 1er embedding: 1.0
Normes de tous: [1.0000001 1.        1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 33/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 34/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 35/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 36/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 37/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 38/421...
```

```
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 39/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.9999999 ]
Norme moyenne: 1.0
BATCH 40/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 41/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 42/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 43/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 44/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 45/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 46/421...
Norme 1er embedding: 1.0000001192092896
Normes de tous: [1.         0.9999999  0.9999999  0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 47/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 48/421...
Norme 1er embedding: 0.9999999403953552
```

```
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 49/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 50/421...
Norme 1er embedding: 1.0
Normes de tous: [1.0000001 1.        1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 51/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 52/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.9999999  1.        ]
Norme moyenne: 1.0
BATCH 53/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 54/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 55/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.9999999  1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 56/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.9999999  0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 57/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 58/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1. 1. 1. 1. 1.]
```

```
Norme moyenne: 1.0
BATCH 59/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 60/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 61/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 62/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 63/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 64/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 65/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 66/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 67/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 68/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
```

```
BATCH 69/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 70/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 71/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 72/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 73/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 74/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 75/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.9999999  0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 76/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 77/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 78/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 79/421...
```

```
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 80/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 81/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 82/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 83/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 84/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 85/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 86/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.9999999  1.         0.9999999 ]
Norme moyenne: 1.0
BATCH 87/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 88/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.9999999  0.99999994 0.99999994 0.9999999 ]
Norme moyenne: 1.0
BATCH 89/421...
Norme 1er embedding: 0.9999999403953552
```

```
Normes de tous: [0.99999994 0.99999994 0.9999999  1.         1.        ]
Norme moyenne: 1.0
BATCH 90/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 91/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.0000001  0.99999994]
Norme moyenne: 1.0
BATCH 92/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 93/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 94/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 95/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.9999999  1.         1.        ]
Norme moyenne: 1.0
BATCH 96/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 97/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 98/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 99/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.9999999  1.         1.         0.99999994]
```

```
Norme moyenne: 1.0
BATCH 100/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.9999999 1.        1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 101/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 102/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 103/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 104/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 105/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 106/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 107/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.9999999  1.         1.        ]
Norme moyenne: 1.0
BATCH 108/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 109/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
```

```
BATCH 110/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 111/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 112/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 113/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         0.9999999 ]
Norme moyenne: 1.0
BATCH 114/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.0000001 ]
Norme moyenne: 1.0
BATCH 115/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 116/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 117/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 118/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.0000001  0.99999994]
Norme moyenne: 1.0
BATCH 119/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.0000001  0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 120/421...
```

```
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 121/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 122/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        0.9999999 1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 123/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 124/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.99999994 0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 125/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 126/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 127/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 128/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.0000001  0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 129/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 130/421...
Norme 1er embedding: 0.9999999403953552
```

```
Normes de tous: [0.99999994 1.         1.0000001  1.         0.99999994]
Norme moyenne: 1.0
BATCH 131/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 132/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        1.        1.        0.9999999 1.       ]
Norme moyenne: 1.0
BATCH 133/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 134/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.9999999 ]
Norme moyenne: 1.0
BATCH 135/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 136/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 137/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 138/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 139/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.9999999  0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 140/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
```

```
Norme moyenne: 1.0
BATCH 141/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 142/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 143/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.0000001  0.99999994]
Norme moyenne: 1.0
BATCH 144/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 145/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.9999999  0.99999994 1.0000001  1.        ]
Norme moyenne: 1.0
BATCH 146/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 147/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 148/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 149/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 150/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
```

```
BATCH 151/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 152/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.9999999  0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 153/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 154/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 1.         1.0000001 ]
Norme moyenne: 1.0
BATCH 155/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 156/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 157/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 158/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 159/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.0000001  1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 160/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 161/421...
```

```
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 162/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 163/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        1.        0.9999999 0.9999999 1.       ]
Norme moyenne: 1.0
BATCH 164/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 165/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 166/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 167/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 168/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 169/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 1.0000001  1.0000001 ]
Norme moyenne: 1.0
BATCH 170/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.9999999  0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 171/421...
Norme 1er embedding: 1.0
```

```
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 172/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.9999999  0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 173/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         1.         0.9999999  0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 174/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 175/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        1.        1.0000001 1.        1.       ]
Norme moyenne: 1.0
BATCH 176/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 177/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 178/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 179/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 180/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 181/421...
Norme 1er embedding: 1.0000001192092896
Normes de tous: [1.0000001  0.99999994 1.         0.99999994 1.        ]
```

```
Norme moyenne: 1.0
BATCH 182/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.9999999  1.         1.        ]
Norme moyenne: 1.0
BATCH 183/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 1.         1.0000001  0.99999994]
Norme moyenne: 1.0
BATCH 184/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 185/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 186/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 187/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 188/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 189/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 190/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.9999999 1.        1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 191/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
```

```
BATCH 192/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 193/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.0000001 ]
Norme moyenne: 1.0
BATCH 194/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.0000001  0.99999994]
Norme moyenne: 1.0
BATCH 195/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 196/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 197/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 198/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 199/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 200/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 201/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 202/421...
```

```
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.9999999  1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 203/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 204/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 205/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 206/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 207/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 208/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 209/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.0000001  1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 210/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 211/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 212/421...
Norme 1er embedding: 0.9999999403953552
```

```
Normes de tous: [0.99999994 1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 213/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 214/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 215/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 1.         1.         0.9999999 ]
Norme moyenne: 1.0
BATCH 216/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 217/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 218/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.0000001  0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 219/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 220/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 221/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 222/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
```

```
Norme moyenne: 1.0
BATCH 223/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 224/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 225/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.9999999  0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 226/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 227/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 228/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 229/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 230/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 231/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 232/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
```

```
BATCH 233/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 234/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 235/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 236/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.9999999 ]
Norme moyenne: 1.0
BATCH 237/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.0000001 ]
Norme moyenne: 1.0
BATCH 238/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 239/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 240/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 241/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.0000001  1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 242/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 243/421...
```

```
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 244/421...
Norme 1er embedding: 1.0000001192092896
Normes de tous: [1.0000001 1.        1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 245/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 246/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 247/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 248/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 249/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 250/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 251/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 252/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 253/421...
Norme 1er embedding: 0.9999999403953552
```

```
Normes de tous: [0.99999994 0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 254/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 255/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 256/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 257/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 258/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 259/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.9999999  1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 260/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 261/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 262/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 263/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
```

```
Norme moyenne: 1.0
BATCH 264/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 265/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 266/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 267/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 268/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 269/421...
Norme 1er embedding: 1.0
Normes de tous: [1.0000001  1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 270/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 271/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 272/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        1.        0.9999999 1.        1.       ]
Norme moyenne: 1.0
BATCH 273/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        1.        1.0000001 1.        1.       ]
Norme moyenne: 1.0
```

```
BATCH 274/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 1.         1.0000001  1.0000001 ]
Norme moyenne: 1.0
BATCH 275/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.9999999  1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 276/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 277/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 278/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 1.         1.         1.0000001 ]
Norme moyenne: 1.0
BATCH 279/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 280/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 281/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 282/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 283/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.        1.0000001 1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 284/421...
```

```
Norme 1er embedding: 1.0
Normes de tous: [1.         0.9999999  0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 285/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 286/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.9999999  0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 287/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 288/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 289/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 290/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 291/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 292/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 293/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.9999999  1.         0.9999999  1.        ]
Norme moyenne: 1.0
BATCH 294/421...
Norme 1er embedding: 0.9999998807907104
```

```
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 295/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 296/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        0.9999999 1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 297/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 298/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.9999999  0.9999999  1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 299/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.99999994 1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 300/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 301/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 0.9999999  1.         0.99999994]
Norme moyenne: 1.0
BATCH 302/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 303/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 304/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.9999999  0.99999994 1.        ]
```

```
Norme moyenne: 1.0
BATCH 305/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 306/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        1.        1.        0.9999999 1.       ]
Norme moyenne: 1.0
BATCH 307/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 308/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.9999999 0.9999999 1.        1.        1.       ]
Norme moyenne: 1.0
BATCH 309/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 310/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 311/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 312/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 313/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 314/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
```

```
BATCH 315/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        1.        1.        1.        1.0000001]
Norme moyenne: 1.0
BATCH 316/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 317/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 318/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 319/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 320/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.0000001  1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 321/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 0.99999994 0.9999999  1.        ]
Norme moyenne: 1.0
BATCH 322/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 323/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 324/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 325/421...
```

```
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 326/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 327/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 328/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 329/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.0000001  0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 330/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 331/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 332/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 333/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 334/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 1.         1.0000001 ]
Norme moyenne: 1.0
BATCH 335/421...
Norme 1er embedding: 1.0
```

```
Normes de tous: [1.         0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 336/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 337/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 338/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 339/421...
Norme 1er embedding: 1.0000001192092896
Normes de tous: [1.0000001  1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 340/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 341/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.9999999  1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 342/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.9999999  1.         0.99999994]
Norme moyenne: 1.0
BATCH 343/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 344/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.0000001  0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 345/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
```

```
Norme moyenne: 1.0
BATCH 346/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 347/421...
Norme 1er embedding: 1.0000001192092896
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 348/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.99999994 1.         1.         0.99999994 0.9999999 ]
Norme moyenne: 1.0
BATCH 349/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.9999999  1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 350/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 351/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 352/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 353/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 354/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 355/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         0.99999994]
Norme moyenne: 1.0
```

```
BATCH 356/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 357/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 358/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 359/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.9999999  0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 360/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.        1.0000001 1.0000001 1.        1.       ]
Norme moyenne: 1.0
BATCH 361/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 362/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 363/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 364/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 365/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 366/421...
```

```
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 367/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 368/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 369/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 370/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 371/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 372/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 373/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 374/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 375/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 376/421...
Norme 1er embedding: 1.0
```

```
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 377/421...
Norme 1er embedding: 1.0
Normes de tous: [1.        1.        1.        1.        1.0000001]
Norme moyenne: 1.0
BATCH 378/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 379/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 380/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 381/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 382/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 383/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 384/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 385/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 386/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.9999998  1.         1.         1.        ]
```

```
Norme moyenne: 1.0
BATCH 387/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 388/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 389/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 390/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 391/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 392/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.9999999  1.0000001  0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 393/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         0.9999999  1.        ]
Norme moyenne: 1.0
BATCH 394/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 395/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 396/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
```

```
BATCH 397/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 398/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 399/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 400/421...
Norme 1er embedding: 0.9999998807907104
Normes de tous: [0.99999994 0.99999994 0.99999994 0.9999999  1.        ]
Norme moyenne: 1.0
BATCH 401/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 402/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 403/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 404/421...
Norme 1er embedding: 1.0
Normes de tous: [1. 1. 1. 1. 1.]
Norme moyenne: 1.0
BATCH 405/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 406/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 407/421...
```

```
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 408/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.9999999  1.         0.99999994 0.99999994]
Norme moyenne: 1.0
BATCH 409/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 410/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [0.99999994 0.99999994 0.99999994 1.         1.0000001 ]
Norme moyenne: 1.0
BATCH 411/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         0.99999994]
Norme moyenne: 1.0
BATCH 412/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 413/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 414/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         0.99999994 1.         1.        ]
Norme moyenne: 1.0
BATCH 415/421...
Norme 1er embedding: 1.0
Normes de tous: [0.99999994 1.         1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 416/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         0.99999994 1.        ]
Norme moyenne: 1.0
BATCH 417/421...
Norme 1er embedding: 1.0
```

```
Normes de tous: [1.         0.99999994 1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 418/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 419/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
BATCH 420/421...
Norme 1er embedding: 1.0
Normes de tous: [1.         0.99999994 1.         1.         1.        ]
Norme moyenne: 1.0
BATCH 421/421...
Norme 1er embedding: 0.9999999403953552
Normes de tous: [1.         1.         1.         1.         0.99999994]
Norme moyenne: 1.0
Transfer Learning embeddings database built: 13463 embeddings
Database shape: (13463, 512)

Le système de recherche par similarité avec Transfer Learning est prêt!
 13463 embeddings générés
Shape embeddings: (13463, 512)
Embeddings sauvegardés: ./tl/recipe_embeddings_database_tl.npy
Metadonnées sauvegardés: ./tl/recipe_embeddings_database_metadata_tl.pkl

Prochaines étapes:
1. Tester avec une image aléatoire: retrieval_system_tl.test_with_random_image()
2. Rechercher avec votre propre image: retrieval_system_tl.search_similar_recipes('path/to/image.jpg')
3. Comparer avec les résultats avec le modèle raw
```

# 6. Initialisation du système et visualisations du modèle TL

In [14]:
```python
# Test avec une image aléatoire du dataset
# Si possible, on recharge le système depuis les fichiers sauvegardés => parfait pour préparer le déploiement

import tensorflow as tf
import pickle
```

```python
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.metrics.pairwise import cosine_similarity
from tensorflow.keras.layers import Layer

#  COUCHE L2 NORMALIZATION PERSONNALISÉE (SÉRIALISABLE)
class L2NormalizationLayer(Layer):
    """Couche personnalisée pour normalisation L2 - sérialisable"""

    def __init__(self, axis=1, **kwargs):
        super(L2NormalizationLayer, self).__init__(**kwargs)
        self.axis = axis

    def call(self, inputs):
        return tf.nn.l2_normalize(inputs, axis=self.axis)

    def compute_output_shape(self, input_shape):
        return input_shape

    def get_config(self):
        config = super(L2NormalizationLayer, self).get_config()
        config.update({'axis': self.axis})
        return config

#  COUCHE EXTRACTION TRIPLET PERSONNALISÉE (SÉRIALISABLE)
class ExtractTripletComponent(Layer):
    """Couche pour extraire une composante du triplet"""

    def __init__(self, component_index, **kwargs):
        super(ExtractTripletComponent, self).__init__(**kwargs)
        self.component_index = component_index

    def call(self, inputs):
        return inputs[:, self.component_index]

    def compute_output_shape(self, input_shape):
        # input_shape = (batch_size, 3, height, width, channels)
        return (input_shape[0], input_shape[2], input_shape[3], input_shape[4])
```

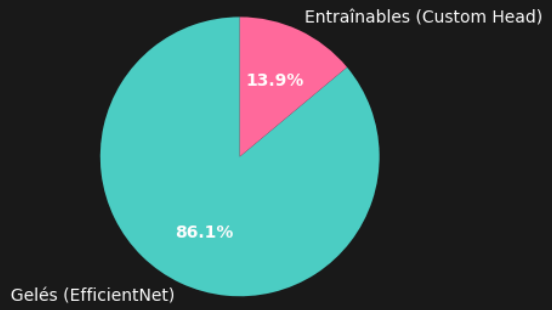
```
    def get_config(self):
        config = super(ExtractTripletComponent, self).get_config()
        config.update({'component_index': self.component_index})
        return config

#  COUCHE STACK TRIPLET PERSONNALISÉE (SÉRIALISABLE)
class TripletStackLayer(Layer):
    """Couche personnalisée pour empiler les embeddings triplet"""

    def __init__(self, **kwargs):
        super(TripletStackLayer, self).__init__(**kwargs)

    def call(self, inputs):
        # inputs = [anchor_emb, positive_emb, negative_emb]
        return tf.stack(inputs, axis=1)

    def compute_output_shape(self, input_shape):
        # input_shape = [(batch_size, embedding_dim), ...]
        batch_size = input_shape[0][0]
        embedding_dim = input_shape[0][1]
        return (batch_size, 3, embedding_dim)

    def get_config(self):
        return super(TripletStackLayer, self).get_config()

CONFIG_STANDALONE = {
    'IMG_SIZE': 224,
}

# On essaie de charger le système sauvegardé
print("Chargement du système sauvegardé...")

try:
    # Chargement bdd embeddings
    embeddings_db = np.load("./tl/recipe_embeddings_database_tl.npy")
    print(f" Embeddings loaded: {embeddings_db.shape}")

    # Chargement métadonnées
    with open("./tl/recipe_embeddings_database_metadata_tl.pkl", 'rb') as f:
        metadata = pickle.load(f)
```

```
    print(" Metadata loaded")

    # Chargement DataFrame
    recipes_df_path = "./data/recipes_with_images_dataframe.pkl"
    if os.path.exists(recipes_df_path):
        recipes_with_images = pd.read_pickle(recipes_df_path)
        print(f" Recipes DataFrame loaded: {len(recipes_with_images)} recipes")
    else:
        print(" recipes_with_images_dataframe_tl.pkl not found!")
        print(" Please save the DataFrame first using:")
        print("   recipes_with_images.to_pickle('recipes_with_images_dataframe.pkl')")

    # Création du système
    retrieval_system_tl = RecipeImageRetrievalTL()
    retrieval_system_tl.embeddings_db = embeddings_db
    retrieval_system_tl.image_paths = metadata['image_paths']
    retrieval_system_tl.image_to_recipe_map = metadata['image_to_recipe_map']

    print(f"Système chargé avec {len(embeddings_db):,} embeddings!")

except FileNotFoundError as e:
    print(f"Fichier introuvable: {e}")
    print("Veuillez vérifier que ces fichiers existent:")
    print("   - recipe_image_retrieval_model_tl.keras (ou dans le dossier ./models/)")
    print("   - recipe_embeddings_database_tl.npy")
    print("   - recipe_embeddings_database_metadata_tl.pkl")
    print("   - recipes_with_images_dataframe.pkl")

except Exception as e:
    print(f" Error loading system: {e}")
    import traceback
    traceback.print_exc()

print("\nVisualisation de l'architecture du modèle...")
retrieval_system_tl.visualize_model_architecture()
```

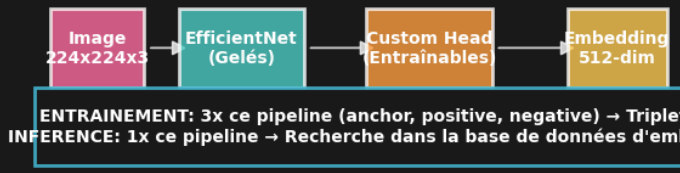
```
Chargement du système sauvegardé...
 Embeddings loaded: (13463, 512)
 Metadata loaded
 Recipes DataFrame loaded: 13463 recipes
Modèle TL chargé: ./tl/best_embedding_recipe_image_retrieval_model_tl.keras
Système chargé avec 13,463 embeddings!

Visualisation de l'architecture du modèle...
```

# Architecture du modèle Transfer Learning

## Distribution des paramètres
## Total: 4,705,443

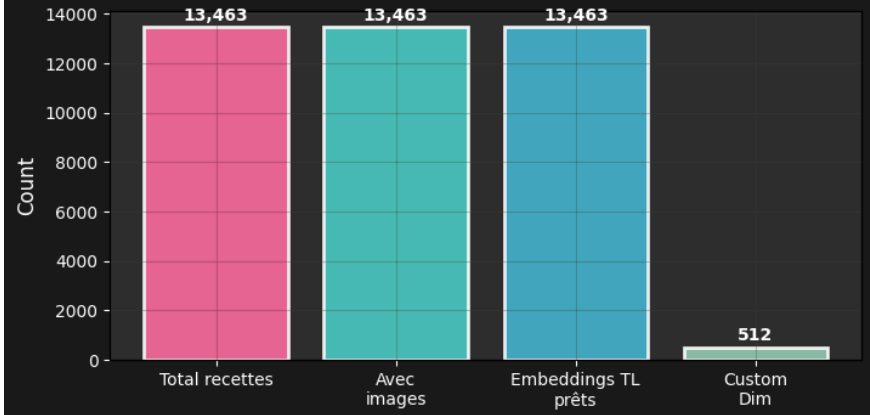



## Architecture et workflow du Transfer Learning




*Même architecture, différent usage: Entraînement vs Inférence*

## Statistiques du Transfer Learning




```
RÉSUMÉ DU MODÈLE DE TRANSFER LEARNING:

Architecture:
• Base: EfficientNetB0 (gélé avec poids ImageNet)
• Custom Head: 1280→1024→512 (entraînée)
• Fonction de loss: Triplet Loss (margin=0.8)
• Régularisation: Dropout (0.3)

Custom Layers (Serializable):
• L2NormalizationLayer: Normalisation L2
• ExtractTripletComponent: Extraction des triplets
• TripletStackLayer: Stack des embeddings pour le loss
• Toutes les couches sont compatibles avec le sauvegarde et le chargement du modèle

Paramètres:
• Total: 4,705,443
• Gelés: 4,049,571 (86.1%)
• Entraînables: 655,872 (13.9%)

Configuration de l'entrainement:
• Epoques: 15 (early stopping à l'époque 9)
• Learning Rate: 0.001 (optimisé pour TL)
• Taille des batch: 32 (bon compromis mémoire/performance)
• Triplet Generator avec hard negative sampling + triplet_margin_accuracy
• Output: 512-dim

Callbacks d'entrainement:
• AdaptiveMarginCallback: Margin adaptatif
• EarlyStopping: Patience 3
• ReduceLROnPlateau: Factor 0.5
• EmbeddingModelCheckpoint: Custom callback
(sauvegarde modèle d'embeddings, pas du modèle de triplets)

Performances attendues:
• Apprentissage de caractéristiques spécifiques aux recettes via triplet loss
• Architecture personnalisée avec couches sérialisables
• Entraînement robuste avec callbacks avancés
• Meilleur modèle époque 6
• triplet_margin_accuracy: 90.7%
• Supérieur au modèle "raw" : compréhension de la sémantique des recettes
```

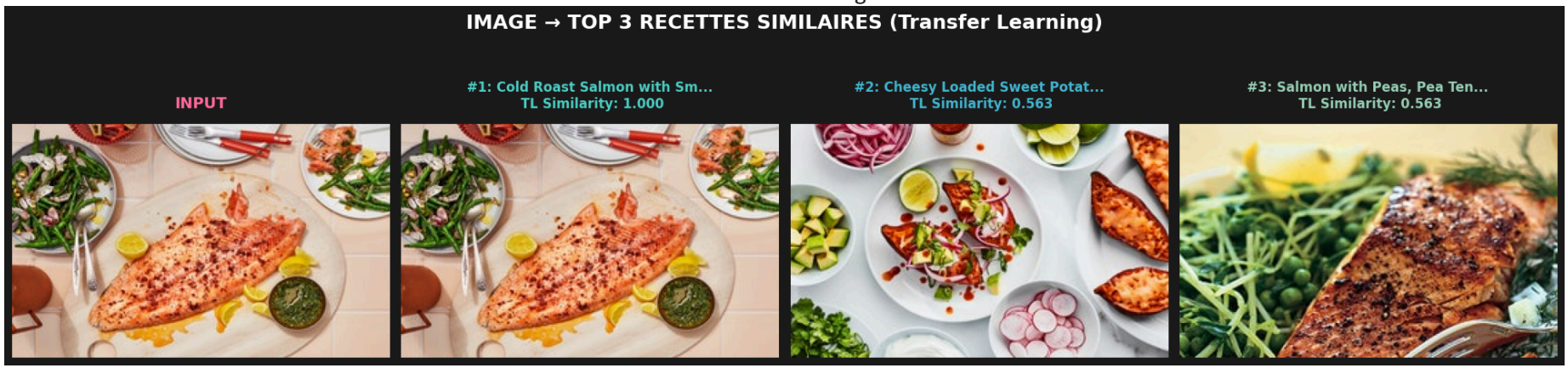
```
Statut:  Prêt pour la recherche par similarité
Dataset: 13,463 recettes, 13,463 avec images
```

```
================================================================================
DÉTAILS DU TRANSFER LEARNING:
================================================================================
- EfficientNetB0 gelé: 4,049,571 paramètres
- Custom Head entrainable: 655,872 paramètres
- Total: 4,705,443 paramètres
- Ratio d'entraînement: 13.9%
- Custom Head: 1280→1024→512 (ENTRAÎNABLES)
```

# 7. Test de recherche sur une image aléatoire du dataset

In [16]:
```
print("\nTest avec une image aléatoire de la base de données...")
retrieval_system_tl.test_with_random_image(top_k=3)
```

```
Test avec une image aléatoire de la base de données...
Test avec l'image: /teamspace/studios/this_studio/.cache/kagglehub/datasets/pes12017000148/food-ingredients-and-recipe-dataset-with-images/versions/1/Food Images/Food Images/cold-roast-salmon-with-smashed-green-bean-salad.jpg
Recherche des 3 recette similaires avec le modèle de Transfer Learning...
```



```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 1.0000
Cold Roast Salmon with Smashed Green Bean Salad
------------------------------------------------------------
Ingredients:
['1 (3 1/2–3 3/4-lb.) whole side of salmon', '7 Tbsp. extra-virgin olive oil, divided, plus more for serving', '4 tsp. kosher s
alt, divided, plus more', '1/2 tsp. crushed red pepper flakes, divided', '1/4 cup fresh lemon juice', '2 lb. green beans, trimm
ed', '1 bunch radishes, preferably French brea...
------------------------------------------------------------
Instructions:
Preheat oven to 300°F. Place salmon on a rimmed baking sheet and rub 2 Tbsp. oil over each side. Season all over with black pep
per, 2 tsp. kosher salt, and 1/4 tsp. red pepper flakes. Arrange skin side down and roast until a tester, metal skewer, or thin
-bladed knife inserted laterally through salmon flesh meets no resistance, 20–25 minutes. (Fish should be opaque throughout and
just able to flake...
================================================================================

RANG #2: 0.5632
Cheesy Loaded Sweet Potatoes
------------------------------------------------------------
Ingredients:
['3 lb. sweet potatoes (4 large or 6 small), scrubbed, halved lengthwise', '3 Tbsp. extra-virgin olive oil, divided', '1 1/2 ts
p. kosher salt', '1/2 tsp. freshly ground black pepper', '1 1/2 cups shredded Monterey Jack or cheddar cheese', 'Sour cream or
plain yogurt, sliced avocado, thinly sliced ra...
------------------------------------------------------------
Instructions:
Fill a large pot with 1" water. Place potatoes in a steamer basket and lower into pot. Cover pot and bring to a boil. Steam unt
il potatoes are fork-tender, 25–30 minutes.
Place an oven rack about 8" from broiler; preheat broiler. Coat a rimmed baking sheet with 1 Tbsp. oil. Arrange potatoes skin s
ide down on prepared pan. Score or lightly mash flesh with a fork; season all over with salt and peppe...
================================================================================

RANG #3: 0.5631
Salmon with Peas, Pea Tendrils, and Dill-Cucumber Sauce
------------------------------------------------------------
Ingredients:
['2 tablespoons olive oil, divided', '1 2 1/2-pound center-cut wild salmon fillet, skin and pinbones removed', '1/2 cup fresh o
range juice', '1/4 cup fresh lemon juice', '1 teaspoon coarse kosher salt', 'Peas and Pea Tendrils with Lemon Dressing', 'Dill-
Cucumber Sauce']
```

```
------------------------------------------------------------
Instructions:
Brush small rimmed baking sheet with 1 tablespoon oil. Place salmon on prepared baking sheet. Mix orange juice and lemon juice
in small bowl; pour over salmon. Drizzle remaining 1 tablespoon oil over salmon; sprinkle with coarse kosher salt and pepper. L
et stand 15 minutes.
Preheat broiler. Broil salmon, without turning fish over, until just opaque in center, watching closely and turning baking s
h...
================================================================================
```

# 8. Recherche sur des images inconnues du système

In [17]:
```python
import tensorflow as tf
import numpy as np
import os
import pickle
import pandas as pd

# Custom layers pour TL (nécessaires pour le chargement)
class L2NormalizationLayer(tf.keras.layers.Layer):
    def __init__(self, axis=1, **kwargs):
        super(L2NormalizationLayer, self).__init__(**kwargs)
        self.axis = axis

    def call(self, inputs):
        return tf.nn.l2_normalize(inputs, axis=self.axis)

    def get_config(self):
        config = super(L2NormalizationLayer, self).get_config()
        config.update({'axis': self.axis})
        return config

class ExtractTripletComponent(tf.keras.layers.Layer):
    def __init__(self, component_index, **kwargs):
        super(ExtractTripletComponent, self).__init__(**kwargs)
        self.component_index = component_index

    def call(self, inputs):
        return inputs[:, self.component_index]
```

```
    def get_config(self):
        config = super(ExtractTripletComponent, self).get_config()
        config.update({'component_index': self.component_index})
        return config

class TripletStackLayer(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super(TripletStackLayer, self).__init__(**kwargs)

    def call(self, inputs):
        return tf.stack(inputs, axis=1)

    def get_config(self):
        return super(TripletStackLayer, self).get_config()

print("Chargement du système Transfer Learning...")

try:
    # Chargement bdd embeddings
    embeddings_db = np.load("./tl/recipe_embeddings_database_tl.npy")

    # Chargement métadonnées
    with open("./tl/recipe_embeddings_database_metadata_tl.pkl", 'rb') as f:
        metadata = pickle.load(f)

    # Chargement DataFrame
    recipes_with_images = pd.read_pickle("./data/recipes_with_images_dataframe.pkl")

    # Création du système
    retrieval_system_tl = RecipeImageRetrievalTL()
    retrieval_system_tl.embeddings_db = embeddings_db
    retrieval_system_tl.image_paths = metadata['image_paths']
    retrieval_system_tl.image_to_recipe_map = metadata['image_to_recipe_map']

    print(f"Système chargé avec {len(embeddings_db):,} embeddings!")

except FileNotFoundError:
    print("Aucun système sauvegardé trouvé.")
    retrieval_system_tl = None
except Exception as e:
    print(f"Erreur: {e}")
```

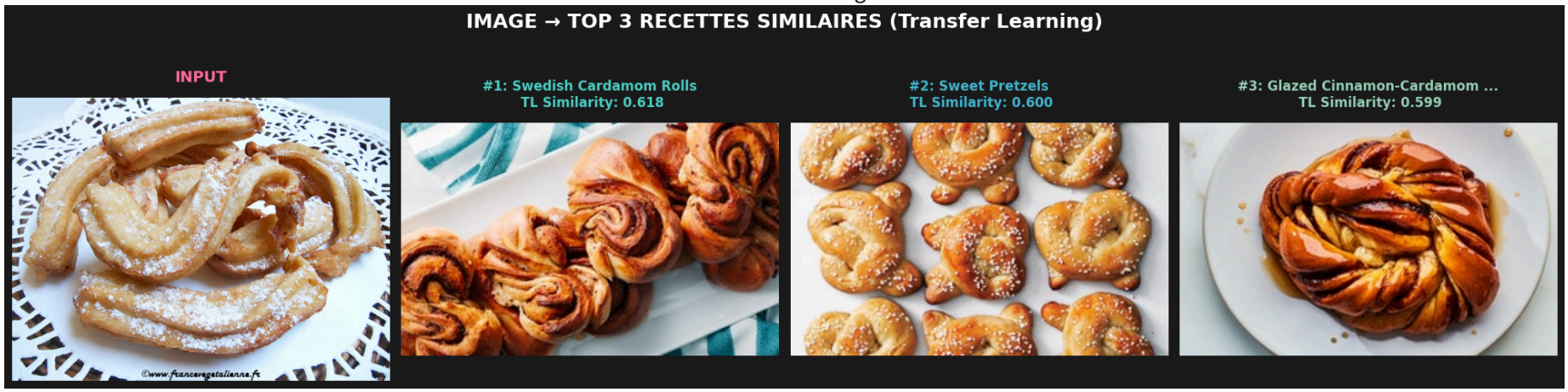
```
    retrieval_system_tl = None

if retrieval_system_tl and retrieval_system_tl.embeddings_db is not None:
    # Test sur les images 1.jpg à 8.jpg
    for i in range(1, 9):
        test_image = f"./test_recipes/{i}.jpg"

        if os.path.exists(test_image):
            print(f"\n{'='*50}")
            print(f"TEST IMAGE {i}.jpg (Transfer Learning)")
            print(f"{'='*50}")

            results = retrieval_system_tl.search_similar_recipes(test_image, top_k=3)
            if results:
                retrieval_system_tl.display_results(test_image, results)
            else:
                print("Aucun résultat trouvé")
        else:
            print(f"Image {i}.jpg introuvable")

else:
    print("Système non disponible")
```

```
Chargement du système Transfer Learning...
Modèle TL chargé: ./tl/best_embedding_recipe_image_retrieval_model_tl.keras
Système chargé avec 13,463 embeddings!

==================================================
TEST IMAGE 1.jpg (Transfer Learning)
==================================================
Recherche des 3 recette similaires avec le modèle de Transfer Learning...
```


IMAGE → TOP 3 RECETTES SIMILAIRES (Transfer Learning)
INPUT
#1: Corn and Jalapeño Chili F...
TL Similarity: 0.604
#2: Chipotle-Glazed Ribs
TL Similarity: 0.604
#3: Bo Ssäm
TL Similarity: 0.601

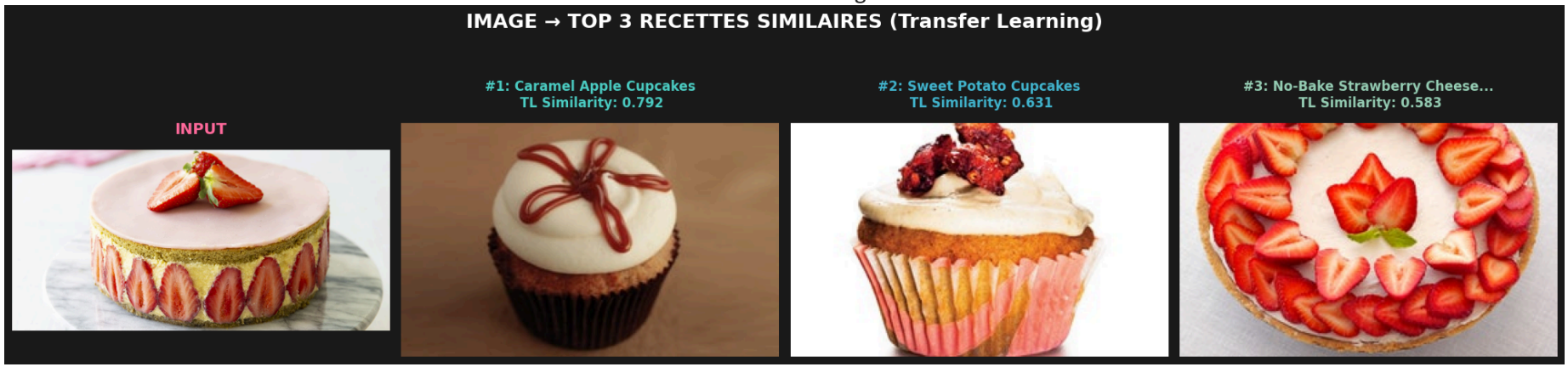
```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 0.6039
Corn and Jalapeño Chili Flatbread
------------------------------------------------------------
Ingredients:
['1 (14-ounce) can creamed corn', '4 1/2 ounces grated haloumi', '4 1/2 ounces grated mozzarella', '1 tablespoon chopped pickle
d jalapeño peppers', '1 tablespoon finely grated lime zest', 'Sea salt and cracked black pepper', 'Cracked black pepper', '4–6
small pita breads, halved', 'Extra-virgin oliv...
------------------------------------------------------------
Instructions:
Preheat a grill pan or grill over high heat. Place the creamed corn, haloumi, mozzarella, jalapeños, lime zest, salt and pepper
in a bowl and mix to combine. Spoon the mixture into the pita halves and press to fill to the edges. Brush the pitas with the o
il and grill for 2–3 minutes on each side or until lightly charred and the cheese has melted. Cut into wedges to serve.
================================================================================

RANG #2: 0.6039
Chipotle-Glazed Ribs
------------------------------------------------------------
Ingredients:
['2 cups white wine vinegar', '1 yellow onion, quartered', '1 head of garlic, halved crosswise', '2 tablespoons table salt', '1
0 1/2 cups water', '4 1/2 pounds (4 racks) American-style pork ribs', 'Lime wedges and pickled jalapeño peppers, to serve', 'Co
rn and Jalapeño Chile Flatbreads, to serve, op...
------------------------------------------------------------
Instructions:
Place the vinegar, onion, garlic, salt and water in a large pot over high heat and bring to a boil. Add the ribs, reduce the he
at to medium and cover with a lid. Simmer for 30–40 minutes or until the ribs are tender.
While the ribs are cooking, make the chipotle glaze. Mix to combine the chipotle chiles, onion flakes, Tabasco, cocoa powder, m
aple syrup, sugar and salt.
Remove the ribs from the coo...
================================================================================

RANG #3: 0.6011
Bo Ssäm
------------------------------------------------------------
Ingredients:
['1 whole 8- to 10-pound bone-in Boston pork butt', '1 cup granulated sugar', '1 cup plus 1 tablespoon kosher salt', '7 tablesp
oons light brown sugar', '1 dozen oysters, shucked', '1 cup Napa Cabbage Kimchi, plus 1 cup puréed', '1 cup Ginger Scallion Sau
ce', 'Ssäm Sauce', '2 cups Short-Grain Rice', ...
```

```
------------------------------------------------------------
Instructions:
1. Put the pork in a roasting pan, ideally one that holds it snugly. Mix together the granulated sugar and 1 cup of the salt in
a bowl, then rub the mixture into the meat; discard any excess salt-and-sugar mixture. Cover the pan with plastic wrap and put
it into the fridge for at least 6 hours, or overnight.
2. Heat the oven to 300°F. Remove the pork from the refrigerator and discard any juices th...
================================================================================

================================================
TEST IMAGE 2.jpg (Transfer Learning)
================================================
Recherche des 3 recette similaires avec le modèle de Transfer Learning...
```



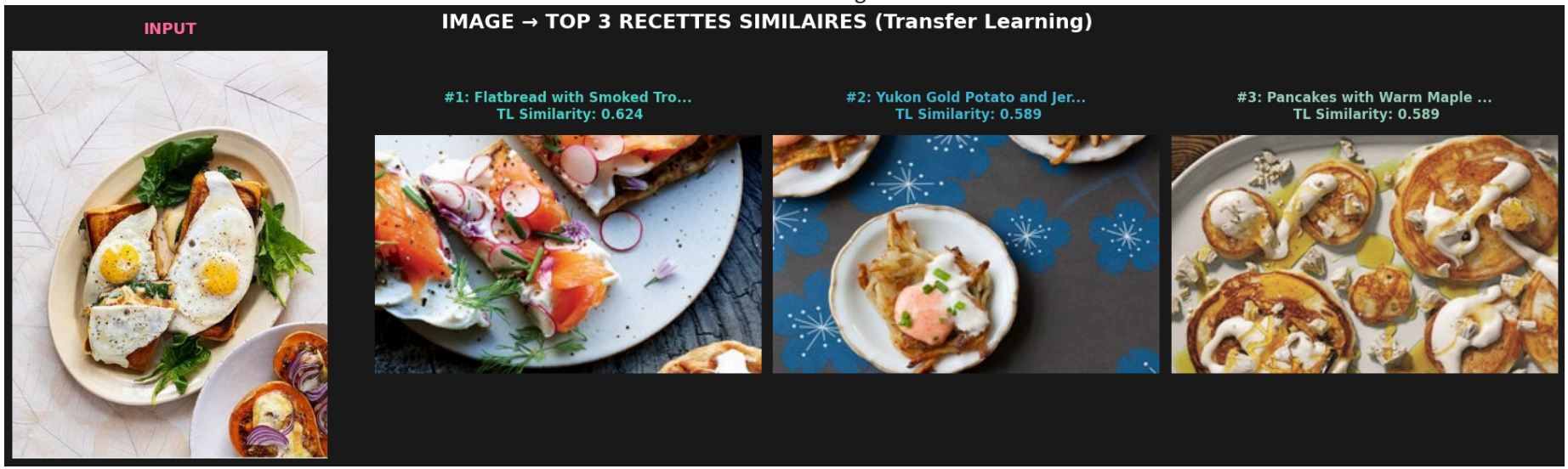
```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 0.6182
Swedish Cardamom Rolls
------------------------------------------------------------
Ingredients:
['1 1/4 cups warm water (105°F.)', '3/4 stick (6 tablespoons) unsalted butter, melted and cooled slightly', '6 tablespoons gran
ulated sugar', 'two 1/4-ounce packages active dry yeast (about 4 1/2 teaspoons total)', '3 large eggs beaten lightly', '1 1/2 t
easpoons salt', '1/4 cup powdered nonfat dry m...
------------------------------------------------------------
Instructions:
In a large bowl combine water, butter, and sugar. Sprinkle yeast over mixture and let stand 5 minutes, or until foamy. Stir in
eggs, salt and dry milk until combined. With a wooden spoon stir in 5 sups flour, 1 cup at a time, and stir mixture until a dou
gh is formed.
On a floured surface, knead dough about 10 minutes, adding enough of the remaining 1 cup flour to make dough smooth and elasti
c. Put...
================================================================================

RANG #2: 0.6000
Sweet Pretzels
------------------------------------------------------------
Ingredients:
['Master Sweet Dough', 'All-purpose flour (for dusting)', '1 large egg white', '6 tablespoons pearl sugar', 'Ingredient info: P
earl sugar, a coarse sugar used for decorating baked goods, is available at specialty foods stores and kingarthurflour.com.']
------------------------------------------------------------
Instructions:
Line 2 large baking sheets with parchment. Punch down dough; divide into 12 equal pieces.
Working with 1 piece at a time and keeping remaining dough covered with a kitchen towel, roll dough on a lightly floured surfac
e into a 17"- long rope. Form rope into a U shape. Lift top of left end; fold over or under opposite side; press together gentl
y about two-thirds down from the top of right side. Lift...
================================================================================

RANG #3: 0.5989
Glazed Cinnamon-Cardamom Buns
------------------------------------------------------------
Ingredients:
['1 cup whole milk', '1 Tbsp. active dry yeast', '1 large egg', '1 large egg yolk', '3 1/2 cups (475 g) all-purpose flour', '1/
2 cup (105 g) granulated sugar', '1 1/2 tsp. (3 g) ground cardamom', '1 tsp. kosher salt', '6 Tbsp. room temperature unsalted b
utter, plus more for bowl', '6 Tbsp. unsalted ...
```

```
------------------------------------------------------------
Instructions:
Heat milk in a small saucepan over low until just warm; an instant-read thermometer should register 105°F–115°F. Pour into the
large bowl of a stand mixer. Whisk in yeast and let sit until foamy, 10–15 minutes. You should see a layer of foam on the surfa
ce; this means that the yeast is active.
Add egg, egg yolk, flour, granulated sugar, cardamom, and salt to yeast mixture and mix with dough hook o...
================================================================================

==================================================
TEST IMAGE 3.jpg (Transfer Learning)
==================================================
Recherche des 3 recette similaires avec le modèle de Transfer Learning...
```



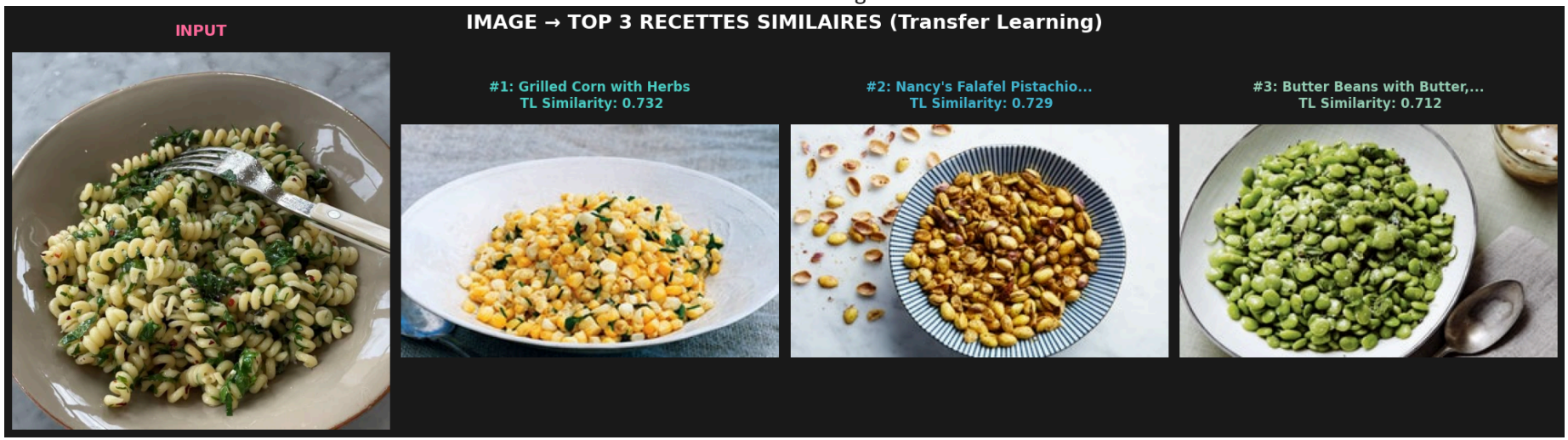
```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 0.7922
Caramel Apple Cupcakes
------------------------------------------------------------
Ingredients:
['2 1/2 cups all-purpose flour', '3 tsp. baking powder', '1 tsp. ground cinnamon', '1/2 tsp. salt', '16 Tbsp. unsalted butter,
at room temperature (European style recommended)', '2 cups sugar', '4 large eggs, at room temperature', '1/3 cup hot water', '2
1/2 cups freshly grated apples (4 to 5 medium...
------------------------------------------------------------
Instructions:
Preheat the oven to 350°F. Line a standard cupcake pan with twelve paper baking cups, and a second pan with six baking cups, or
grease pans with butter if not using baking cups.
Sift together the flour, baking powder, cinnamon, and salt on a sheet of parchment paper or wax paper and set aside.
Place the butter in the bowl of a stand mixer or in a bowl with a handheld electric mixer. Beat on medium...
================================================================================

RANG #2: 0.6311
Sweet Potato Cupcakes
------------------------------------------------------------
Ingredients:
['1/2 cup oat flour', '6 tablespoons all-purpose flour', '1/2 teaspoon baking soda', '6 tablespoons granulated sugar', '2 table
spoons unsalted butter, at room temperature', '1 egg white', '1/4 cup cooked or canned sweet potato', '1/4 teaspoon vanilla ext
ract', '2 tablespoons skim milk', "1/4 cup con...
------------------------------------------------------------
Instructions:
Heat oven to 350°F. Line 16 cups of two 12-cup mini-muffin pans with mini-muffin papers. In a bowl, combine flours and baking s
oda. In another bowl, cream together sugar and butter with an electric mixer on medium. Add egg white, sweet potato and vanill
a; beat on low until well combined. Add flour mixture and milk; beat on low until just combined. Do not overmix. Fill muffin cu
ps 2/3 full. Bake un...
================================================================================

RANG #3: 0.5829
No-Bake Strawberry Cheesecake
------------------------------------------------------------
Ingredients:
['8 ounces cream cheese, softened', '1/3 cup sugar', '1 cup sour cream', '2 teaspoons pure vanilla extract', '8 ounces prepared
whipped topping, thawed', '1 prepared graham cracker crust (6 ounces)', '1 pound fresh strawberries, hulled and halved lengthwi
se']
```

```
------------------------------------------------------------
Instructions:
1. Beat the cream cheese until smooth with an electric mixer. Gradually beat in the sugar. Beat in the sour cream and vanilla u
ntil just combined. Fold in the whipped topping. Scrape mixture into the piecrust. (There may be some fi lling left over. If s
o, reserve it to be decoratively piped onto the top of the pie.) Chill in the refrigerator for 4 hours.
2. Starting in the center, arrange the stra...
================================================================================

================================================
TEST IMAGE 4.jpg (Transfer Learning)
================================================
Recherche des 3 recette similaires avec le modèle de Transfer Learning...
```



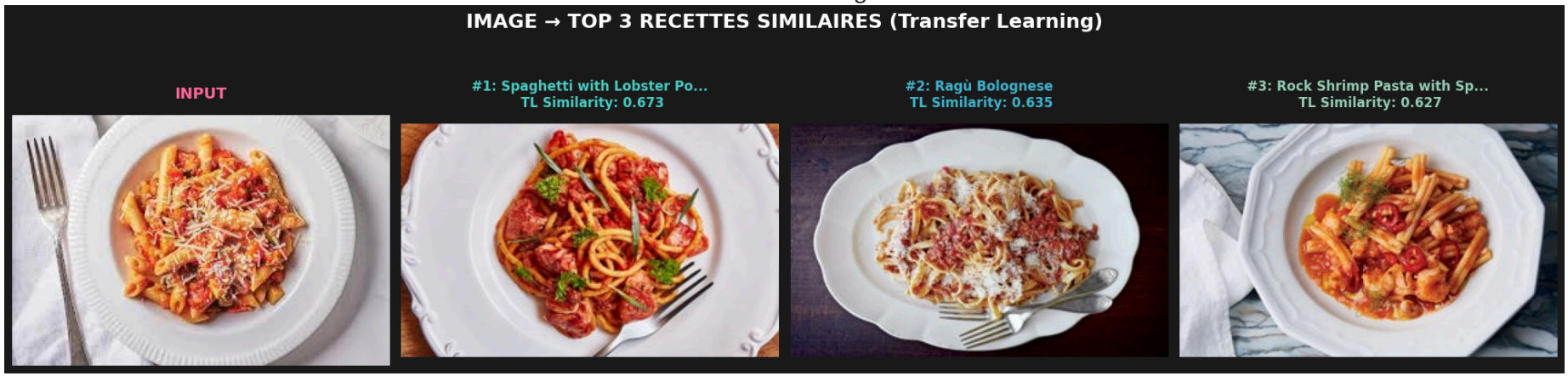
```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 0.6239
Flatbread with Smoked Trout, Radishes, and Herbs
------------------------------------------------------------
Ingredients:
['3 cups plain whole-milk yogurt', 'Vegetable oil (for grill)', '1/2 Garlic-Herb Naan or 1 pound store-bought pizza dough, room
temperature, halved', '2 (5-ounce packages) smoked trout, coarsely flaked', '4 radishes, trimmed, thinly sliced on a mandolin
e', '1/4 cup coarsely chopped dill', '2 tablesp...
------------------------------------------------------------
Instructions:
Line a fine-mesh sieve with cheesecloth and set over a medium bowl. Place yogurt in sieve, cover with plastic wrap, and let dra
in in refrigerator at least 1 day and up to 2 days if you want it slightly thicker. Discard excess liquid. (Or skip this step e
ntirely and use 1 1/2 cups store-bought labneh instead.)
Prepare a grill for medium-high, indirect heat (for a charcoal grill, bank coals on one s...
================================================================================

RANG #2: 0.5892
Yukon Gold Potato and Jerusalem Artichoke Latkes with Apple-Horseradish Mayonnaise and Taramasalata
------------------------------------------------------------
Ingredients:
['1 (2-inch) piece fresh horseradish root, peeled and finely grated', '1/2 cup unsweetened applesauce', '2 teaspoons apple cide
r vinegar', '1 tablespoon Pommery or Dijon mustard', '1 cup mayonnaise', '3 pounds Yukon gold potatoes, peeled', '1 pound Jerus
alem artichokes, thoroughly washed', '1 large ...
------------------------------------------------------------
Instructions:
In a medium bowl, stir together the horseradish, applesauce, vinegar, mustard, and mayonnaise. Season with salt and pepper. DO
AHEAD: The sauce can be made and stored, in an airtight container in the refrigerator, up to 3 days.
Line a large bowl with a clean cloth napkin or lint-free kitchen towel.
Working in batches, use the larger side of a box grater or a food processor fitted with a grater att...
================================================================================

RANG #3: 0.5892
Pancakes with Warm Maple Syrup & Coffee Butter
------------------------------------------------------------
Ingredients:
["2 cups high-quality store-bought pancake mix (such as Robby's pancake mix)", '1/4 cup all-purpose flour', '2 cups whole mil
k', '2 large eggs', '4 tablespoons unsalted butter, melted', '1/2 cup brewed coffee', '6 tablespoons unsalted butter, softene
d', 'Pure maple syrup, warmed, for serving', 'Slic...
```

```
------------------------------------------------------------
Instructions:
1. In a medium bowl, whisk the pancake mix, flour, milk, eggs, and melted butter together until smooth. Cover the bowl tightly
with plastic wrap and refrigerate for at least 2 hours, and as long as overnight.
2. Put the coffee in a small saucepan, bring to a simmer over medium heat, and cook until reduced by about half. Remove from th
e heat and cool completely.
3. Put 4 tablespoons of the softened...
================================================================================

================================================
TEST IMAGE 5.jpg (Transfer Learning)
================================================
Recherche des 3 recette similaires avec le modèle de Transfer Learning...
```



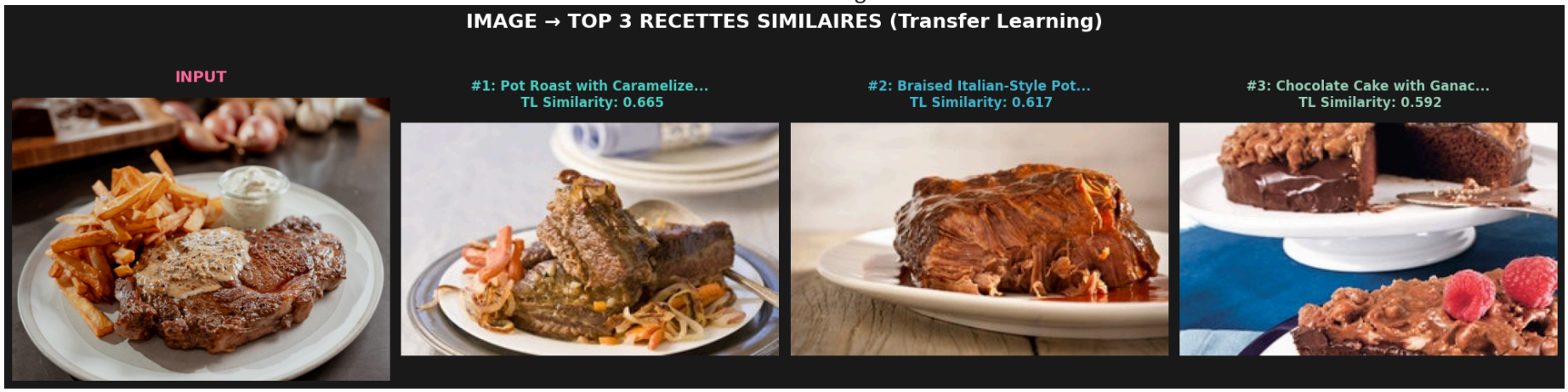
```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 0.7317
Grilled Corn with Herbs
------------------------------------------------------------
Ingredients:
['8 ears of corn in the husk', '1/4 cup chopped mixed fresh herbs such as chives, parsley, basil, sage, and tarragon', '6 table
spoons lime butter sauce']
------------------------------------------------------------
Instructions:
Prepare grill for cooking over medium-hot charcoal (moderate heat for gas).
Grill corn (in husks) on lightly oiled grill rack, turning, covered, until kernels are tender, 20 to 30 minutes. Remove corn fr
om grill and let stand until cool enough to handle but still warm, about 10 minutes.
Discard husks and stem ends from corn. Cut kernels off cobs with a large knife and toss with herbs and lime butt...
================================================================================

RANG #2: 0.7289
Nancy's Falafel Pistachios
------------------------------------------------------------
Ingredients:
['1 tablespoon ground cumin', '1 tablespoon ground coriander', '1 tablespoon dried ground basil', '1 tablespoon dried ground ma
rjoram', '1 tablespoon dried ground rosemary', '2 1/2 teaspoons turmeric', '2 cups (300 g) whole peeled garlic cloves', '2 cups
(475 ml) olive oil', '6 dried hot red chilies...
------------------------------------------------------------
Instructions:
Combine the first six ingredients in a small bowl. Set aside the "falafel" spice.
Line a plate with paper towels, Using a mandolin with a safety attachment to protect your fingers, cut the garlic cloves into p
aper-thin slices. Pour olive oil into a medium-size heavy skillet and heat over medium-high heat. Add the garlic slices and coo
k until crisp and light golden brown; do not burn or the garlic ...
================================================================================

RANG #3: 0.7118
Butter Beans with Butter, Mint, and Lime
------------------------------------------------------------
Ingredients:
['Kosher salt', '6 cups fresh shelled butter beans or frozen baby lima beans', '3 tablespoons unsalted butter, cut into small p
ieces', 'Juice of 2 large limes', '1 cup loosely packed mint leaves, chopped', 'Freshly ground black pepper', '1 teaspoon grate
d lime zest, for garnish']
------------------------------------------------------------
```

```
Instructions:
In a medium saucepan, bring 6 cups water and 1 tablespoon salt to a boil over high heat. Add butter beans and cook until tende
r, 9-12 minutes, depending on the size of the beans. Drain in a colander; shake colander several times to shed as much water as
possible.
Put butter in a large serving bowl, and pour warm butter beans on top. Toss beans with butter until all the butter is melted. A
dd lime j...
================================================================================

================================================
TEST IMAGE 6.jpg (Transfer Learning)
================================================
Recherche des 3 recette similaires avec le modèle de Transfer Learning...
```



```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 0.6735
Spaghetti with Lobster Pomodoro
------------------------------------------------------------
Ingredients:
['Kosher salt', '2 (1 1/4-lb.) live lobsters', '2 Tbsp. extra-virgin olive oil', '2 Tbsp. ghee or unsalted butter', '1/4 small
red onion, thinly sliced', '3 garlic cloves, thinly sliced', '2 sprigs basil', '1 (14-oz.) can whole peeled San Marzano tomatoe
s', '12 oz. spaghetti', '4 oz. nduja, broken i...
------------------------------------------------------------
Instructions:
Bring a large pot of salted water to a rolling boil. Working one at a time, cook lobsters 3 minutes, then transfer to a large b
owl of ice water. Let cool just until you can comfortably handle them, about 1 minute, then twist off claws where the knuckles
meet the body and return them to pot of boiling water. Cook 2 minutes (leave bodies in ice water). Add claws back to ice water
and let both claws ...
================================================================================

RANG #2: 0.6347
Ragù Bolognese
------------------------------------------------------------
Ingredients:
['2 tablespoons butter', '2 tablespoons extra-virgin olive oil', '1 medium yellow onion, peeled and finely chopped', '2 small c
elery ribs, finely diced', '1 carrot, peeled and finely diced', '2 ounces prosciutto di Parma, finely chopped', '¾ pound ground
beef chuck', '¾ pound ground pork', 'Salt and...
------------------------------------------------------------
Instructions:
Heat the butter and oil together in a heavy large pot over medium heat. Add the onions, celery, and carrots and cook, stirring
often with a wooden spoon, until the vegetables have softened and the onions are translucent, 5–10 minutes. Stir in the prosciu
tto. Add the ground chuck and pork, season to taste with salt and pepper, and cook, breaking up the clumps of meat with the bac
k of the spoon, unt...
================================================================================

RANG #3: 0.6272
Rock Shrimp Pasta with Spicy Tomato Sauce
------------------------------------------------------------
Ingredients:
['1 (28-ounce) can whole peeled tomatoes, preferably San Marzano, drained', '1/3 cup olive oil, plus more for drizzling', '1/2
medium fennel bulb, fronds reserved, core removed, bulb thinly sliced', '8 garlic cloves, smashed', '1 Fresno chile, very thinl
y sliced, divided', '1/4 cup dry white wine', ...
```

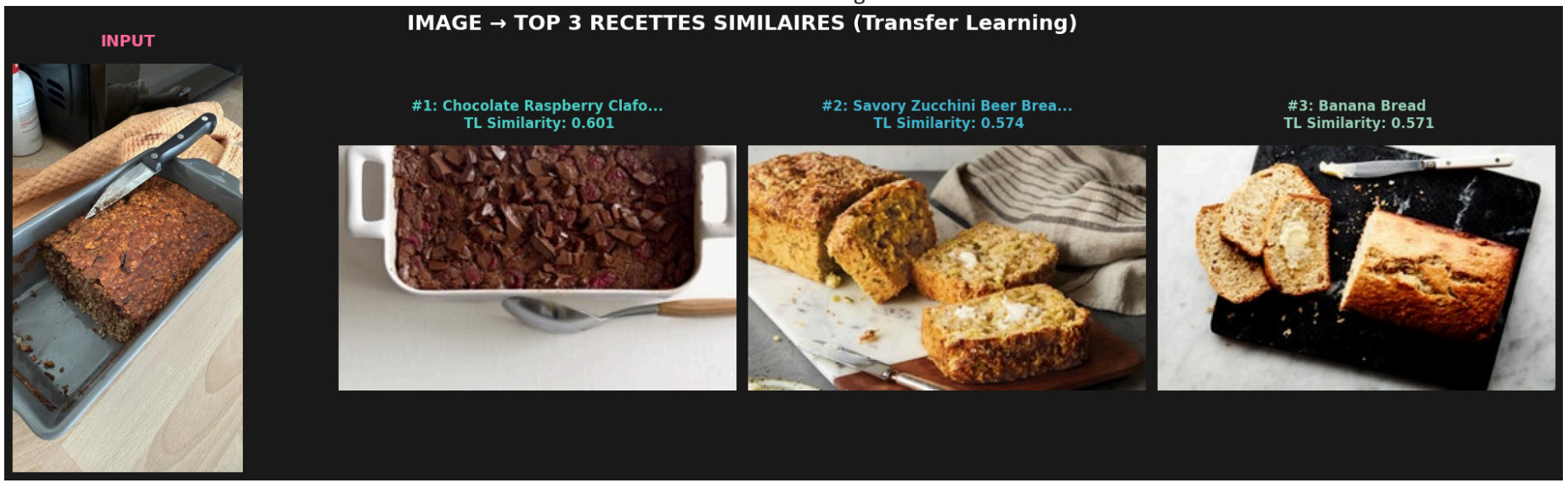
```
------------------------------------------------------------
Instructions:
Set a fine-mesh sieve over a medium bowl. Working over sieve, squeeze tomatoes to release juices and break up flesh. Let tomato
es drain in sieve, collecting juices in bowl, until ready to use.
Heat 1/3 cup oil in a large Dutch oven or other heavy pot over medium. Cook fennel, garlic, and half of chile, stirring often,
until garlic is golden and fennel is starting to brown around the edges, 5–8 min...
================================================================================

================================================
TEST IMAGE 7.jpg (Transfer Learning)
================================================
Recherche des 3 recette similaires avec le modèle de Transfer Learning...
```



```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 0.6649
Pot Roast with Caramelized Onions and Roasted Carrots
------------------------------------------------------------
Ingredients:
['1/2 cup canola oil', 'Kosher salt and freshly ground black pepper', '5 pounds boneless short ribs, denuded (all surface fat r
emoved; have your butcher do this)', '1 cup dry sherry', '4 carrots, peeled and roughly chopped', '2 large onions, peeled and r
oughly chopped', '8 stalks celery, peeled and ...
------------------------------------------------------------
Instructions:
Position racks in upper and lower thirds of oven and preheat to 350°F. Season beef liberally with salt and pepper. In large Dut
ch oven or heavy ovenproof pot over moderately high heat, heat oil until hot but not smoking. Add beef and sear until dark brow
n and crisp on both sides, about 10 minutes total. Transfer beef to large plate. Pour off oil in pan and discard. Add sherry an
d simmer uncovered,...
================================================================================

RANG #2: 0.6167
Braised Italian-Style Pot Roast
------------------------------------------------------------
Ingredients:
['One 5-inch sprig fresh thyme', '5 fresh Italian, flat leafed parsley stems', '2 dried bay leaves or 1 fresh bay leaf', 'One 5
-inch sprig fresh rosemary', '2 juniper berries, crushed', 'One 2-pound piece shoulder of beef, bottom round, or pot roast', 'K
osher salt and freshly ground black pepper', '...
------------------------------------------------------------
Instructions:
Combine all the ingredients in the center of a piece of cheesecloth that is large enough to hold the herb sprigs, and tie in a
bundle with butcher's string.
1. Preheat the oven to 350°F.
2. Season the beef with salt and pepper, then lightly dust with flour. Melt the butter in a large (6-quart) flameproof casserol
e set over medium-high heat. When it is foaming, add the beef and brown it on all side...
================================================================================

RANG #3: 0.5924
Chocolate Cake with Ganache and Praline Topping
------------------------------------------------------------
Ingredients:
['Nonstick vegetable oil spray', '1/2 cup (1 stick) unsalted butter, cut into 1/2-inch cubes, room temperature', '1/4 cup natur
al unsweetened cocoa powder', '1/2 cup boiling water', '1 cup all purpose flour', '1 cup sugar', '1/2 teaspoon baking soda',
```

'1/4 teaspoon coarse kosher salt', '1/4 cup butt...
---
Instructions:
Preheat oven to 350°F. Spray 9-inch cake pan with nonstick spray. Line with parchment. Spray parchment. Dust with flour, tappin g out any excess.
Place butter and cocoa powder in medium bowl. Pour 1/2 cup boiling water over; stir. Let stand 2 minutes; whisk until blended. Whisk flour, sugar, baking soda, and coarse salt in another medium bowl. Whisk buttermilk, egg, and vanilla in large bowl. Grad u...
==========

==========
TEST IMAGE 8.jpg (Transfer Learning)
==========
Recherche des 3 recette similaires avec le modèle de Transfer Learning...



```
================================================================================
RECETTES SIMILAIRES (Transfer Learning)
================================================================================

RANG #1: 0.6013
Chocolate Raspberry Clafoutis
------------------------------------------------------------
Ingredients:
['12 ounces fresh raspberries (2 3/4 cups)', '1 tablespoon granulated sugar', '1 cup whole milk', '1/2 stick unsalted butter, m
elted', '3 large eggs', '1/2 cup packed dark brown sugar', '1/3 cup all-purpose flour', '2 tablespoons unsweetened cocoa powde
r', '1/4 teaspoon salt', '3 to 3 1/2 ounces bit...
------------------------------------------------------------
Instructions:
Preheat oven to 400°F with rack in middle. Butter a 1 1/2-quarts shallow baking dish.
Toss berries with granulated sugar and let stand 15 minutes.
Blend milk, butter, eggs, brown sugar, flour, cocoa, and salt in a blender until smooth. Scatter berries (with juices) evenly i
n baking dish, then pour batter over top.
Bake until slightly puffed and firm to the touch, about 35 minutes. Remove from oven...
================================================================================

RANG #2: 0.5737
Savory Zucchini Beer Bread
------------------------------------------------------------
Ingredients:
['2 medium zucchini or yellow squash (about 12 ounces total), coarsely grated', '1 teaspoon salt, divided', '¼ cup plus 1 table
spoon olive oil, plus more for greasing', '3 cups all-purpose flour (substitute up to half whole-wheat flour)', '1 tablespoon b
aking powder', '1 tablespoon sugar', '½ teaspo...
------------------------------------------------------------
Instructions:
Toss the zucchini with 1/2 teaspoon of the salt in a colander and let stand for 15 minutes. Gather it up in your hands and sque
eze out as much moisture as you can.
Preheat the oven 375°F. Generously grease an 8 x 4-inch loaf pan.
Whisk together the flour, baking powder, remaining 1/2 teaspoon salt, the sugar, and black pepper. Add the zucchini, scallions,
coarsely shredded cheese, and 1/4 cup of t...
================================================================================

RANG #3: 0.5709
Banana Bread
------------------------------------------------------------
Ingredients:
['1 stick butter, melted, plus softened butter for greasing', '2 cups flour', '1/2 teaspoon salt', '1 1/2 teaspoons baking powd
```

er', '1 cup sugar', '3 very ripe bananas, mashed with a fork until smooth', '2 eggs', '1 teaspoon vanilla extract', '1/2 cup ch
opped walnuts (optional)', '1/2 cup shredded u...
---
Instructions:
Heat the oven to 350°F. Grease a 9x5-inch loaf pan with softened butter.
Whisk together the flour, salt, baking powder, and sugar in a large bowl.
Mix together the melted butter and mashed bananas in a separate bowl. Beat in the eggs and vanilla until well combined. Stir th
is mixture into the dry ingredients just enough to combine everything. Gently fold in the nuts and coconut if you’re using th
e...
==========