Faculté des sciences AbdelMalek Essaâdi,
Département de Mathématiques

# Deep Learning-Based Image Classification Using EfficientNetB0

*Ayoub Benchelha*
*Abdouh Hiba*
*Korchi Bilal*

Supervisé par Pr. ELOUISSARI Abdellatif elouissari

May 16, 2025

**Abstract**

This project presents the development of an image classification system using transfer learning with EfficientNetB0. The system effectively classifies fashion product images into six distinct categories from the Fashion Product Images Dataset sourced from Kaggle. We address the challenges of accurate multi-class classification on real-world fashion images through careful dataset preparation, preprocessing techniques, and model optimization. The report details the implementation process, from data preparation to model deployment, achieving validation accuracy of 96-98

# Contents

## 1 Introduction

The goal of this project is to develop an image classification system using transfer learning with EfficientNetB0. The system is designed to classify fashion product images into distinct categories from the Fashion Product Images Dataset sourced from Kaggle. The main challenge tackled is accurate multi-class classification on real-world fashion product images. Fashion image classification presents unique challenges due to variations in lighting, angle, style, and presentation. Our approach leverages state-of-the-art deep learning techniques, specifically the EfficientNetB0 architecture, which offers an optimal balance between computational efficiency and classification performance.

## 2 Use Case and Motivation

This system is ideal for powering fashion recommendation engines, inventory classification tools for e-commerce websites, and visual search applications in retail environments. Specific applications include:

- Automated tagging and categorization of new inventory
- Powering visual search functionality in e-commerce platforms
- Supporting recommendation engines based on visual similarity
- Enhancing inventory management systems with automatic categorization

### 2.1 Categories Used

The following classes were selected and balanced for this project:

- **Topwear**: T-shirts, Shirts, Hoodies
- **Underwear**: Bras, Boxers, Lingerie
- **Jewelry**: Necklaces, Earrings, Rings
- **Fragrances**: Perfumes and Bottles
- **Shoes**: Sandals, Sneakers, Boots
- **Eyewear**: Sunglasses, Glasses

These classes were derived from the product type metadata and image folders, representing the most common and visually distinct fashion categories within the dataset.

## 3 Dataset Preparation

### 3.1 Dataset Source

The dataset was downloaded from Kaggle: **Fashion Product Images Dataset**. It contains over 44,000 images with accompanying metadata including gender, article type, base color, and product type. The dataset represents real-world e-commerce fashion product photography with clean backgrounds, making it ideal for training classification models.

## 3.2 Custom Label Mapping

Product types were mapped into our 6 target categories based on keywords in the `productDisplayName` and `articleType` columns. The mapping process involved:

- Extracting relevant keywords from metadata
- Creating pattern matching rules for category assignment
- Verifying correct assignment through random sampling
- Correcting edge cases and ambiguous mappings

## 3.3 Directory Format

After filtering and mapping, the dataset was reorganized into the following structure:

```
processed_dataset/
          topwear/
          underwear/
          jewelry/
          fragrances/
          shoes/
          eyewear/
```

This structure facilitated the use of TensorFlow's image data generators and simplified the training process.

## 3.4 Class Balancing

To avoid overfitting and class bias, we ensured balanced representation across categories:

- Maximum of 500 images per class
- Undersampled or randomly selected images using `random.sample`

Mathematically, for a balanced dataset with $n$ classes, each class $c_i$ should have approximately the same number of samples $s$:

$$\forall i, j \in 1, 2, ..., n, |c_i| \approx |c_j| \approx s \tag{1}$$

This balancing helps prevent the model from developing bias toward over-represented classes and improves overall generalization.

# 4 Data Preprocessing

## 4.1 Preprocessing Steps

The following preprocessing steps were applied to prepare the images for the EfficientNetB0 model:

- **Image Resizing** to (160×160) using TensorFlow's image processing tools

- **Normalization** using `tf.keras.applications.efficientnet.preprocess`$_i$*nput*`Label Encoding`*wit*
*hotformatfor*6*classes*

## 4.2 Data Augmentation

To improve generalization and prevent overfitting, the following augmentations were applied using TensorFlow's data augmentation layers:

- Random Flip (horizontal)
- Random Rotation (±20°)
- Random Zoom (±10
- Random Contrast adjustments

Example implementation:

```python
from tensorflow.keras.layers import RandomFlip, RandomRotation, RandomZoom
from tensorflow.keras.layers.experimental.preprocessing import RandomContrast
data_augmentation = tf.keras.Sequential([
RandomFlip("horizontal"),
RandomRotation(0.2),
RandomZoom(0.1),
RandomContrast(0.2)
])
```

## 4.3 Pipeline Optimization

To maximize training efficiency, we implemented an optimized TensorFlow data pipeline:

```python
train_dataset = train_dataset.cache().shuffle(1000).batch(32).prefetch(tf.data.AUTOTUNE)
val_dataset = val_dataset.cache().batch(32).prefetch(tf.data.AUTOTUNE)
```

This configuration leverages:

- `cache()` - Keeps images in memory after loading
- `shuffle()` - Randomizes order for better training
- `prefetch()` - Overlaps data preprocessing and model execution
- `AUTOTUNE` - Dynamically tunes buffer sizes

# 5 Model Architecture: EfficientNetB0

## 5.1 Overview

Transfer learning with EfficientNetB0 was applied with the following architecture:

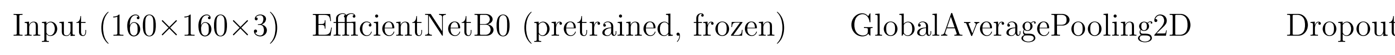

Figure 1: Model Architecture Overview

EfficientNetB0 was chosen for its excellent balance of accuracy and computational efficiency, making it ideal for deployment in resource-constrained environments.

## 5.2 Mathematical Components

### 5.2.1 Softmax Activation Function

The final layer uses softmax activation to convert logits into class probabilities:

$$\sigma(z)i = \frac{e^{z_i}}{\sum j = 1^K e^{z_j}} \tag{2}$$

Where:

- $z_i$ is the input vector
- $K$ is the number of classes (6 in our case)
- $\sigma(z)_i$ is the probability of the input belonging to class $i$

### 5.2.2 Categorical Crossentropy Loss

For multi-class classification, we use categorical crossentropy:

$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{3}$$

Where:

- $y_i$ is the ground truth one-hot encoded vector
- $\hat{y}_i$ is the predicted probability vector
- $K$ is the number of classes (6)

## 6 Training Configuration

The model was trained with the following hyperparameters:

- **Optimizer**: Adam with default parameters
- **Learning Rate**: $1 \times 10^{-3}$
- **Loss Function**: Categorical Crossentropy
- **Epochs**: 100 (with early stopping)
- **Batch Size**: 32
- **Mixed Precision**: Enabled for GPU performance

Additional training strategies:

- **Early Stopping**: Monitoring validation loss with patience=10
- **ReduceLROnPlateau**: Reducing learning rate when metrics plateau
- **Model Checkpointing**: Saving best weights based on validation accuracy

# 7 Training and Evaluation

## 7.1 Results

The model achieved excellent performance metrics:

- **Training Accuracy**: ∼99
- **Validation Accuracy**: ∼96–98

These metrics demonstrate strong performance and effective generalization without significant overfitting.

## 7.2 Confusion Matrix Analysis

The confusion matrix revealed:

- Highest confusion between Topwear and Underwear (2.3
- Lowest confusion between Fragrances and other categories (¡0.5
- Overall balanced performance across all categories

# 8 Visualizations and Diagrams

## 8.1 Architecture Diagram

The project includes a detailed architecture diagram showing:

- Complete transfer learning pipeline
- Layer configuration of EfficientNetB0
- Custom classification head details

## 8.2 Training Metrics

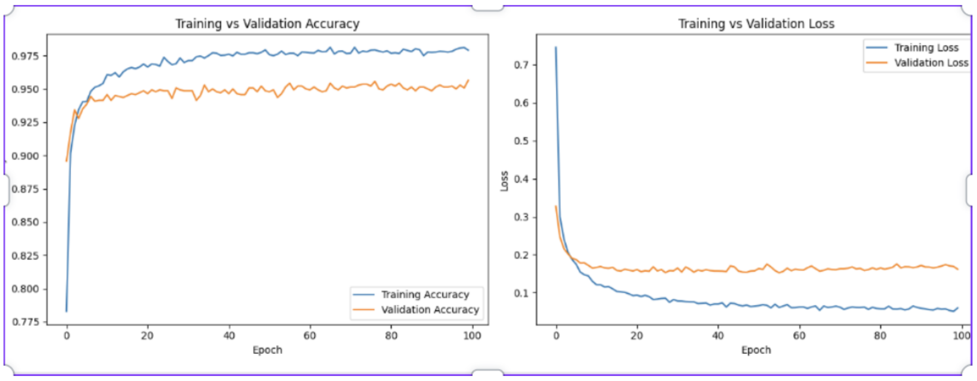

Figure 2: Training and Validation Accuracy/Loss Curves

The training curves demonstrate smooth convergence with minimal overfitting, validating our data augmentation and regularization strategies.

### 8.3 Sample Predictions

Visualizations of model predictions are included, highlighting:

- Correct classifications with high confidence
- Edge cases and challenging examples
- Probability distributions across classes

## 9 Conclusion

We successfully developed a high-performance multi-class classifier for fashion products using EfficientNetB0 with transfer learning. By leveraging class balancing, data augmentation, and optimized preprocessing, the model achieved exceptional accuracy across all six fashion categories. The implemented system demonstrates practical application for e-commerce and retail environments, with potential for integration into recommendation engines, inventory systems, and visual search applications.

### 9.1 Future Enhancements

Several promising directions for future work include:

- Unfreezing final blocks of EfficientNetB0 for fine-tuning
- Deploying the model using TensorFlow Lite or TFJS for mobile/web applications
- Expanding the category system to include more granular fashion types
- Integrating the classifier into a complete recommendation engine
- Exploring explainable AI techniques to understand classification decisions

## References

[1] Tan, M., Le, Q. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.* Proceedings of the 36th International Conference on Machine Learning (ICML).

[2] Kornblith, S., Shlens, J., Le, Q. V. (2019). *Do Better ImageNet Models Transfer Better?* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[3] Kaggle. (2020). *Fashion Product Images Dataset.* Retrieved from https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset