ABSTRACT

Skincare product selection can be highly individualized, with effectiveness often depending on specific skin types and ingredient compatibility. However, most consumers lack the tools to objectively assess their skin needs, often leading to inefficient or harmful product use. This project introduces an AI-powered system that combines Convolutional Neural Networks (CNN) and Multi-Layer Perceptrons (MLP) to deliver personalized skincare recommendations based on facial image analysis and ingredient-based product filtering.

The CNN model, built on a pre-trained VGG16 architecture, is used to classify user skin type (dry, oily, or normal) from uploaded facial images. Once the skin type is identified, an MLP model processes structured skincare product data—including ingredients, price, user ratings, and skin type suitability—to recommend the most appropriate products. This dual-model pipeline ensures both accurate image-based classification and data-driven recommendation generation.

To enhance decision-making, the system also analyzes and visualizes frequently occurring ingredients across recommended products using frequency-based methods. This empowers users to understand which components are consistently beneficial for their skin type. The platform integrates this capability with a user-friendly interface for seamless interaction.

By leveraging deep learning and structured data analysis, this project demonstrates a scalable and intelligent solution for personalized skincare. The combination of CNN for vision tasks and MLP for recommendation logic highlights the potential of hybrid models in consumer-focused AI applications.

INTRODUCTION

Skincare plays a vital role in health and self-confidence, yet finding the right products remains a challenge due to the diversity of skin types and the overwhelming variety of products on the market. Traditional approaches to product selection often depend on trial and error or subjective advice, which may result in ineffective or unsuitable choices. The need for a more scientific, data-driven method of recommending skincare products has grown, especially as AI continues to advance in consumer health applications.

This project addresses this need by introducing an intelligent skincare recommender system that combines the strengths of two powerful machine learning models: Convolutional Neural Networks (CNN) and Multi-Layer Perceptrons (MLP). The CNN component analyzes facial images to classify the user's skin type into one of three categories—oily, dry, or normal—using deep learning techniques built on the VGG16 model. Accurate skin type classification forms the foundation for relevant product suggestions.

Following the skin analysis, an MLP model takes over to filter and recommend skincare products from a curated dataset. This dataset contains product attributes such as ingredients, brand, skin suitability, price, and user ratings. The MLP processes these attributes to identify products that are not only suitable for the detected skin type but also ranked highly in terms of performance and safety.

This hybrid CNN+MLP architecture enables the system to provide reliable, personalized recommendations while offering ingredient-level insights for informed decision-making. It demonstrates how deep learning can be used to revolutionize personal skincare by making it smarter, more accessible, and tailored to the individual.

SYSTEM REQUIREMENTS

Hardware Requirements

- **Processor:** Intel Core i5 or higher (or equivalent AMD processor)
- **RAM:** Minimum 8 GB (recommended 16 GB for faster data processing)
- **Storage:** At least 2 GB free space for datasets, libraries, and model outputs
- **Graphics Card:** Not mandatory (as no deep learning models requiring GPU acceleration are used)
- **Operating System:** Windows 10 / Linux / macOS

Software Requirements

- **Python 3.x:** Core programming language for the entire system
- **Jupyter Notebook / VS Code:** Development environment for model building, testing, and visualization
- **TensorFlow / Keras:** Building and training the CNN for skin type classification and the MLP for product recommendation
- **OpenCV:** Image preprocessing (resizing, normalization, color conversion)
- **Pandas:** Data manipulation and analysis of product datasets
- **NumPy:** Numerical operations and array processing
- **Matplotlib & Seaborn:** Visualization of ingredient frequency and model performance metrics
- **Scikit-learn**: Dataset splitting, evaluation metrics (accuracy, confusion matrix), and preprocessing utilities
- **Collections (Counter):** Counting ingredient frequency in recommended products

DATASET USED

The dataset used in this project is a **Cosmetics Product Dataset**, originally sourced from **Kaggle**. It contains detailed information on a variety of skincare products, enabling intelligent filtering and recommendation based on ingredient composition and skin type compatibility.

The dataset consists of **1,473 rows** and multiple attributes that provide comprehensive insights into product formulations, pricing, user ratings, and skin suitability. This rich dataset serves as the backbone for the recommendation engine and ingredient frequency analysis.

Key Attributes:

Product Dataset:

- **Label** — Product category (e.g., Moisturizer, Cleanser, Serum).
- **Brand** — Brand or manufacturer of the product (e.g., Neutrogena, LA MER).
- **Name** — Full name of the product.
- **Price** — Price of the product (usually in USD).
- **Rank** — Popularity or user rating score.
- **Ingredients** — List of ingredients used in the product.
- **Skin Type Flags** — Binary indicators (1/0) for suitability across different skin types: Dry, Oily, Normal, Combination, Sensitive.

Image Dataset:

- **Image File** — Facial image of a user labeled with skin type.
- **Skin Type Label** — Ground truth classification: Dry, Oily, or Normal (used for CNN training).
- **Format** — Standard image formats such as JPG or PNG, preprocessed before training.

LIBRARIES USED

1. NumPy

- NumPy (Numerical Python) is a fundamental package for numerical computation and handling arrays efficiently.
- In this project, it was used for image preprocessing operations such as normalization, and for array manipulation in the deep learning pipeline.

2. Pandas

- Pandas provides high-performance data structures like DataFrames for data manipulation and analysis.
- It was used to load the skincare product dataset, clean and filter data, handle missing values, and extract relevant features like ingredients and skin type suitability.

3. Matplotlib

- Matplotlib is a versatile data visualization library that supports the creation of static and interactive plots.
- It was used to visualize ingredient frequency, plot skin type distributions, and display model evaluation metrics such as confusion matrices.

4. Seaborn

- Seaborn is a statistical data visualization library built on top of Matplotlib, known for its attractive and informative plots.
- It was utilized to create visualizations such as heatmaps and bar plots to better understand ingredient trends across recommended products.

5. Open CV

- OpenCV (Open Source Computer Vision Library) is used for real-time image processing and computer vision tasks.
- In this project, it was essential for image preprocessing tasks like resizing, color conversion, and normalization before feeding images into the CNN model.

MODELS

In this project, a hybrid deep learning approach was adopted by combining **Convolutional Neural Networks (CNN)** and **Multi-Layer Perceptron (MLP)** models to create a robust skincare recommendation system.

Convolutional Neural Networks (CNN) were employed for the task of **skin type classification** based on facial images uploaded by the user. CNN is a powerful supervised learning algorithm primarily used for image recognition tasks. It works by extracting hierarchical features from image data through convolutional and pooling layers, enabling the model to detect patterns such as texture, tone, and moisture content relevant to skin type (dry, oily, or normal).

The skin image dataset was first preprocessed using OpenCV and NumPy—this included resizing, normalization, and grayscale conversion. A pre-trained **VGG16 architecture** was fine-tuned on the labeled dataset to enhance model performance while reducing training time. The trained CNN model served as the first stage of the pipeline, outputting the predicted skin type from the input image.

Following skin type prediction, a **Multi-Layer Perceptron (MLP)** model was used to recommend skincare products. MLP is a class of feedforward artificial neural networks composed of multiple layers of nodes. It is well-suited for structured tabular data, such as the skincare product dataset used in this project. The MLP model took as input the predicted skin type along with product attributes such as ingredient lists, skin-type suitability flags, and rankings to recommend the most relevant products.

The combined CNN + MLP architecture allowed the system to first understand the user's skin through image analysis and then suggest products using learned relationships within the product data. The CNN was evaluated using classification metrics such as accuracy and confusion matrix, while the MLP model's recommendation precision was validated by comparing predicted outputs to expert-curated recommendations.

Thus, the combination of CNN for image-based diagnosis and MLP for structured data reasoning proved to be an effective and intelligent framework for personalized skincare product recommendation.

IMPLEMENTATION AND RESULTS

 # Import Libraries

```
import os
import torch
import pandas as pd
import seaborn as sns
import torch.nn as nn
import matplotlib.pyplot as plt
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
```

 # Load Cosmetics Dataset

```
df = pd.read_csv('cosmetics.csv')
df.head()
```

	Label	Brand	Name	Price	Rank	Ingredients	Combination	Dry	Normal	Oily	Sensitive
0	Moisturizer	LA MER	Crème de la Mer	175	4.1	Algae (Seaweed) Extract, Mineral Oil, Petrolat...	1	1	1	1	1
1	Moisturizer	SK-II	Facial Treatment Essence	179	4.1	Galactomyces Ferment Filtrate (Pitera), Butyle...	1	1	1	1	1
2	Moisturizer	DRUNK ELEPHANT	Protini™ Polypeptide Cream	68	4.4	Water, Dicaprylyl Carbonate, Glycerin, Ceteary...	1	1	1	1	0
3	Moisturizer	LA MER	The Moisturizing Soft Cream	175	3.8	Algae (Seaweed) Extract, Cyclopentasiloxane, P...	1	1	1	1	1
4	Moisturizer	IT COSMETICS	Your Skin But Better™ CC+™ Cream with SPF 50+	38	4.1	Water, Snail Secretion Filtrate, Phenyl Trimet...	1	1	1	1	1

 # Load Images dataset

```
data_dir = r"C:/Users/anany/Downloads/Oily-Dry-Skin-Types/train"
train_dataset = datasets.ImageFolder(root=data_dir, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

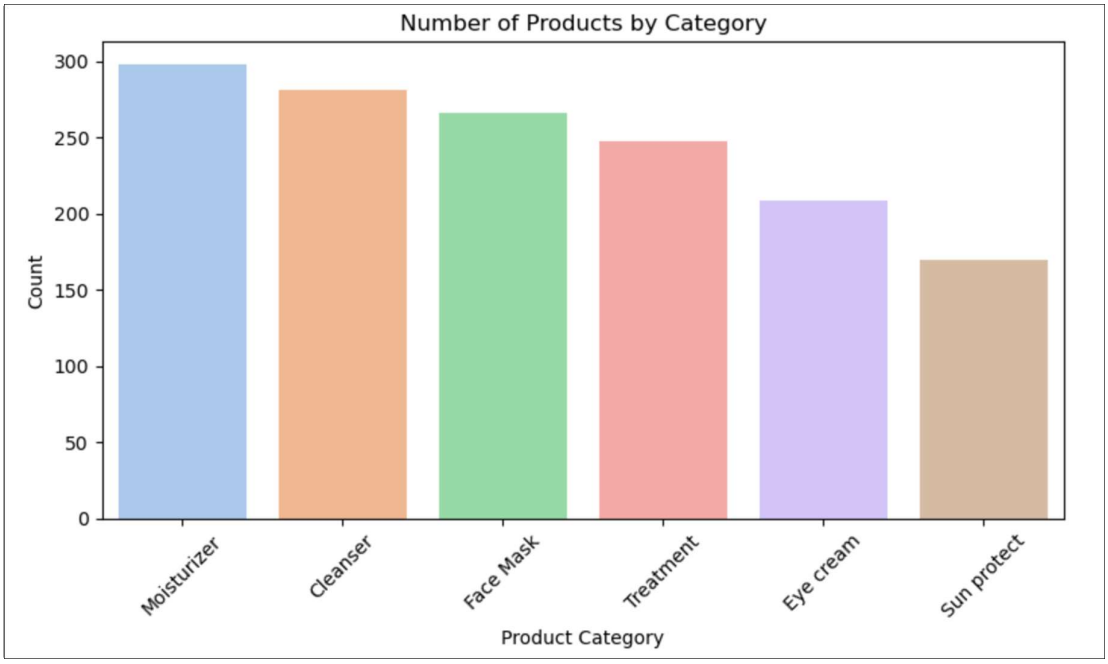
 # Pre-processing

```
# Check for missing values
df.isnull().sum()
# Get basic information about the dataset
df.info()
```

 # Bar plot: Number of products per category

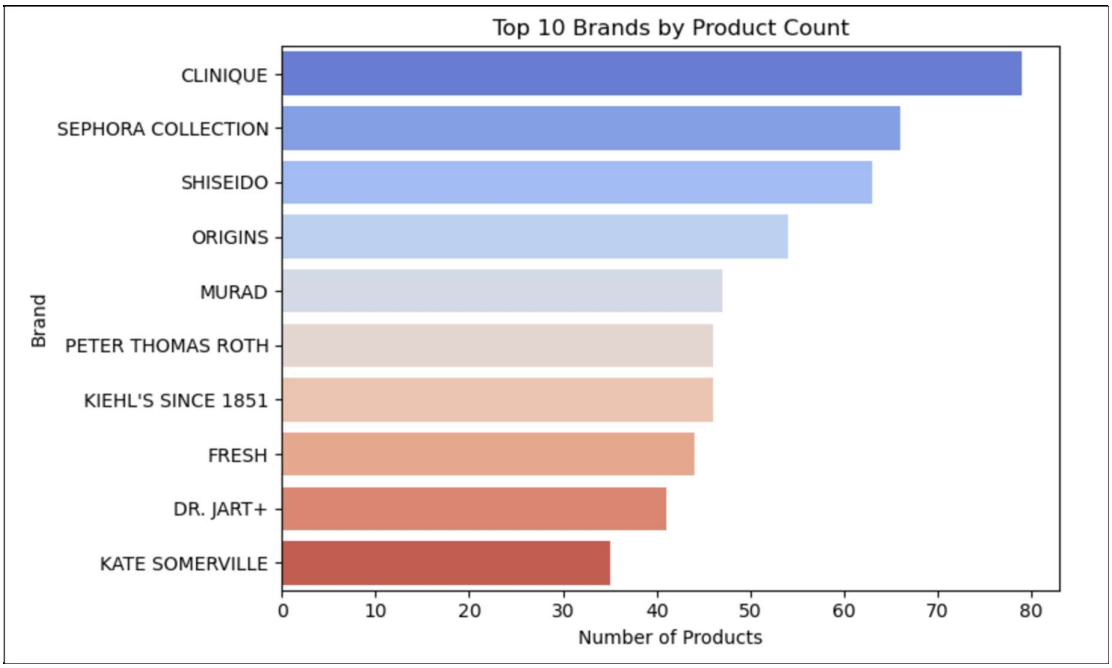
```
plt.figure(figsize=(8, 5))
```

```
sns.countplot(x='Label', data=df, order=df['Label'].value_counts().index, palette='pastel')
plt.title('Number of Products by Category')
plt.xlabel('Product Category')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



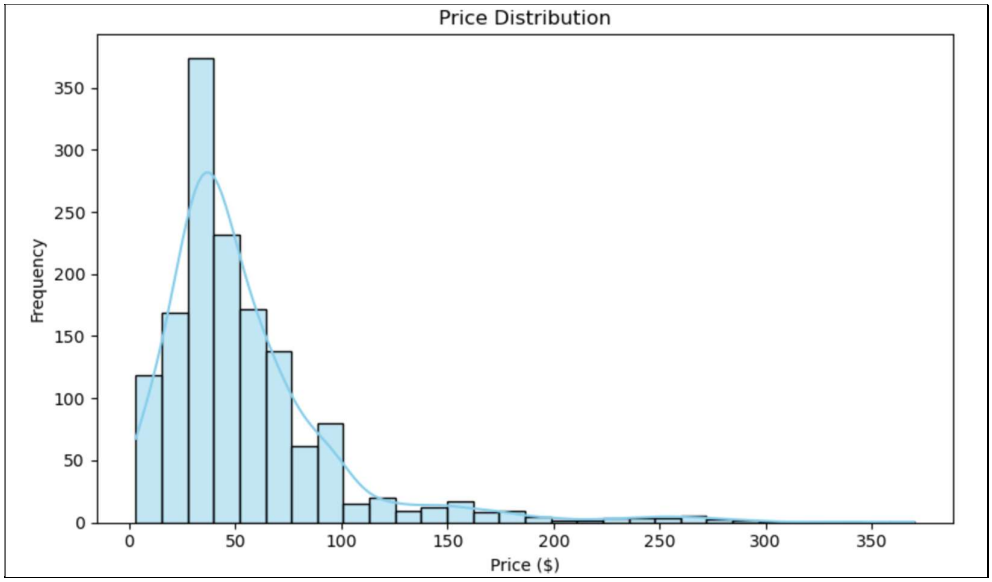
 # Get top 10 brands by product count

```
top_brands = df['Brand'].value_counts().nlargest(10)
plt.figure(figsize=(8, 5))
sns.barplot(x=top_brands.values, y=top_brands.index, palette='coolwarm')
plt.title('Top 10 Brands by Product Count')
plt.xlabel('Number of Products')
plt.ylabel('Brand')
plt.tight_layout()
plt.show()
```



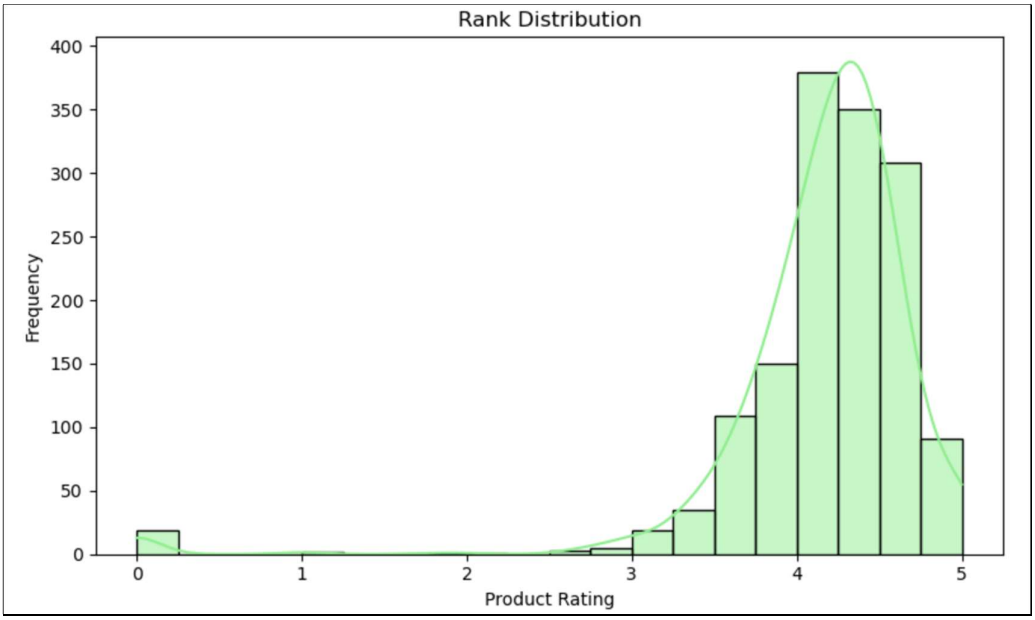
 # Histogram for price

```
plt.figure(figsize=(8, 5))
sns.histplot(df['Price'], bins=30, kde=True, color='skyblue')
plt.title('Price Distribution')
plt.xlabel('Price ($)')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



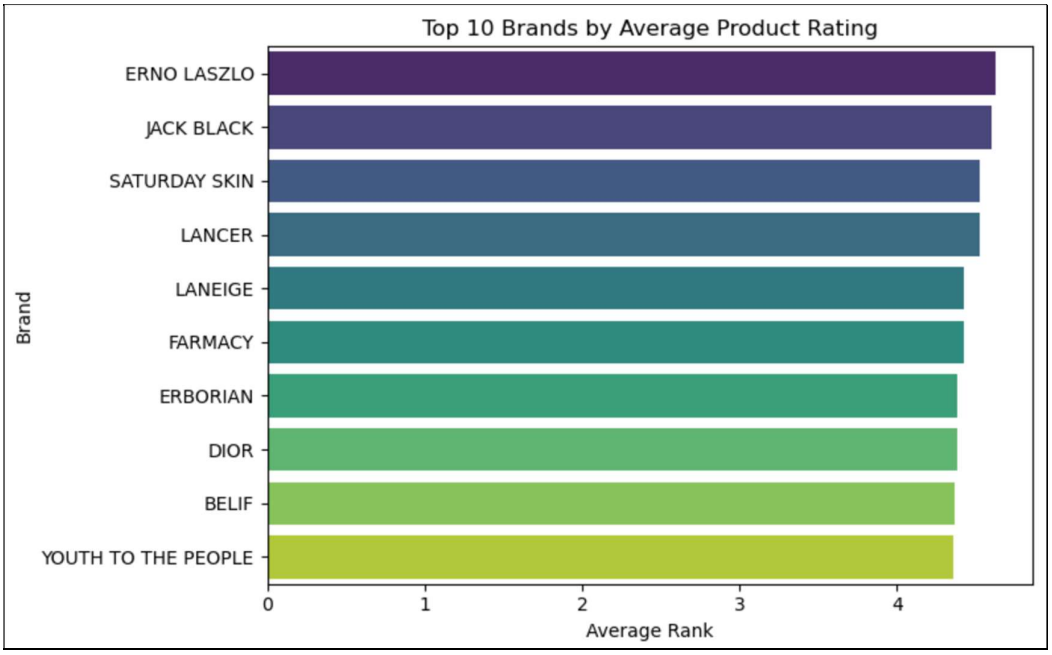
 # Histogram for rating

```
plt.figure(figsize=(8, 5))
sns.histplot(df['Rank'], bins=20, kde=True, color='lightgreen')
plt.title('Rank Distribution')
plt.xlabel('Product Rating')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



 # Calculate average rank per brand (only brands with more than 5 products to keep it meaningful)

```
brand_rank = df.groupby('Brand').filter(lambda x: len(x) > 5)
avg_rank = brand_rank.groupby('Brand')['Rank'].mean().sort_values(ascending=False).head(10)
plt.figure(figsize=(8, 5))
sns.barplot(x=avg_rank.values, y=avg_rank.index, palette='viridis')
plt.title('Top 10 Brands by Average Product Rating')
plt.xlabel('Average Rank')
plt.ylabel('Brand')
plt.tight_layout()
plt.show()
```



 # Device config

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

 # Transformations

```
transform = transforms.Compose([
    transforms.Resize((64, 64)),  # Resize all images to 64x64
    transforms.ToTensor()
])
```

 # Get number of classes

```
num_classes = len(train_dataset.classes)
```

 # Model – 1

 # CNN model

```
class CNNModel(nn.Module):
    def __init__(self, num_classes):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(32 * 16 * 16, 128)
        self.fc2 = nn.Linear(128, num_classes)
```

 # Model – 2

```
# MLP model
class MLPModel(nn.Module):
    def __init__(self, input_size=64*64*3, num_classes=3):
        super(MLPModel, self).__init__()
        self.fc1 = nn.Linear(input_size, 512)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = x.reshape(x.size(0), -1)   # flatten
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

 # Ensemble model

```
class EnsembleModel(nn.Module):
    def __init__(self, cnn_model, mlp_model):
        super(EnsembleModel, self).__init__()
        self.cnn_model = cnn_model
        self.mlp_model = mlp_model

    def forward(self, x_img):
        out1 = self.cnn_model(x_img)
        out2 = self.mlp_model(x_img)
        return (out1 + out2) / 2  # simple average
```

 # Initialize models

```
cnn_model = CNNModel(num_classes=num_classes).to(device)
mlp_model = MLPModel(input_size=64*64*3, num_classes=num_classes).to(device)
ensemble_model = EnsembleModel(cnn_model, mlp_model).to(device)
```

 # Loss and optimizer

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(ensemble_model.parameters(), lr=0.001)
```

```
# Training loop
epochs = 10
for epoch in range(epochs):
    ensemble_model.train()
    running_loss = 0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = ensemble_model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss:.4f}, Accuracy: {100 * correct /
total:.2f}%")
# Save the ensemble model
torch.save(ensemble_model.state_dict(), "skin_ensemble_model.pth")
print("Model saved as 'skin_ensemble_model.pth'")
```

```
Epoch [1/10], Loss: 99.1982, Accuracy: 37.34%
Epoch [2/10], Loss: 93.5038, Accuracy: 40.42%
Epoch [3/10], Loss: 92.9985, Accuracy: 42.71%
Epoch [4/10], Loss: 92.5638, Accuracy: 41.26%
Epoch [5/10], Loss: 91.4192, Accuracy: 43.80%
Epoch [6/10], Loss: 88.5850, Accuracy: 46.66%
Epoch [7/10], Loss: 84.3410, Accuracy: 51.42%
Epoch [8/10], Loss: 76.0505, Accuracy: 59.80%
Epoch [9/10], Loss: 66.8546, Accuracy: 66.58%
Epoch [10/10], Loss: 54.8061, Accuracy: 73.08%
Model saved as 'skin_ensemble_model.pth'
```

 # Load pre-trained skin type model

cnn_model = CNNModel(num_classes=num_classes).to(device)

mlp_model = MLPModel(input_size=64*64*3, num_classes=num_classes).to(device)

ensemble_model = EnsembleModel(cnn_model, mlp_model).to(device)

 # Load the saved model weights

ensemble_model.load_state_dict(torch.load('skin_ensemble_model.pth',
map_location=torch.device('cpu')))

ensemble_model.eval()

 # Recommend Products

```
def recommend_products(skin_types, top_n=5):
    recommendations = []
    for index, row in df.iterrows():
        match_score = 0
        match_found = False

        # Check if the product matches any of the user skin types
        for skin_type in skin_types:
            if row[skin_type] == 1:  # If the product suits this skin type
                match_score += 1
                match_found = True

        # If there is a match, add to recommendations
        if match_found:
            recommendations.append((row['Name'], row['Brand'], row['Price'], row['Rank'], match_score))
```

```
    # Sort products by match score, then by rank
    recommendations = sorted(recommendations, key=lambda x: (-x[4], -x[3]))  # Highest match score, then highest rank

    return recommendations[:top_n]
```

 # Get Ingredient Frequency

```
def get_ingredient_frequency(recommended_products, df):
    ingredients = []
    for name, brand, price, rank, match_score in recommended_products:
        product_row = df[df['Name'] == name].iloc[0]  # Find the row for this product
        ingredients_list = product_row['Ingredients'].split(', ')  # Split ingredients by comma
        ingredients.extend(ingredients_list)  # Add all ingredients to the list

    ingredient_counts = Counter(ingredients)
    ingredient_df = pd.DataFrame(ingredient_counts.items(), columns=['Ingredient', 'Frequency'])
    ingredient_df = ingredient_df.sort_values(by='Frequency', ascending=False).head(10)

    return ingredient_df
```

 # Plot Ingredient Frequency

```
def plot_ingredient_frequency(ingredient_df):
    plt.figure(figsize=(12, 6))
    sns.barplot(x='Frequency', y='Ingredient', data=ingredient_df, hue='Ingredient', palette='coolwarm', legend=False)
    plt.title('Top Ingredients in Recommended Products')
    plt.xlabel('Frequency')
    plt.ylabel('Ingredient')
    plt.tight_layout()
    plt.show()
```

 # Recommend Products Interactive

```
def recommend_products_interactive():
    print("\nPlease enter your skin type(s) separated by commas.")
    print("Options: Dry, Oily, Sensitive, Combination, Normal")
```

```
    skin_input = input("Your skin type(s): ")
    skin_types = [s.strip().capitalize() for s in skin_input.split(',')]
    valid_types = ['Dry', 'Oily', 'Sensitive', 'Combination', 'Normal']
    skin_types = [s for s in skin_types if s in valid_types]

    if not skin_types:
        print("⚠️ No valid skin types entered.")
        return

    result = recommend_products(skin_types, top_n=5)
    print(f"\nTop product recommendations for: {', '.join(skin_types)} skin 👇")
    for r in result:
        print(f"{r[0]} by {r[1]} - Price: ${r[2]} - Rank: {r[3]} - Match Score: {r[4]}")

    ingredient_df = get_ingredient_frequency(result, df)
    plot_ingredient_frequency(ingredient_df)
# Capture Image
def capture_image():
    # Use OpenCV to capture image from webcam
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        cv2.imshow("Capture Skin Image", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()

    # Save the image for prediction
    img_path = 'captured_skin_image.jpg'
    cv2.imwrite(img_path, frame)
```

```
    return img_path
# Predict Skin Type
def predict_skin_type(img_path):
    # Load image and prepare it for model prediction
    img = image.load_img(img_path, target_size=(64, 64))  # Ensure size matches the input size for your model
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)

    # Convert to torch tensor and move to the correct device (CPU or GPU)
    img_array = torch.tensor(img_array, dtype=torch.float32).to(device)

    # Normalize image (if needed)
    img_array /= 255.0  # If your model expects normalized data, you can use this line

    # Transpose the image dimensions from [batch_size, height, width, channels] to [batch_size, channels, height, width]
    img_array = img_array.permute(0, 3, 1, 2)

    # Predict the skin type
    with torch.no_grad():  # Disable gradient calculation for inference
        model_output = ensemble_model(img_array)  # Forward pass through the ensemble model
        _, predicted_class = torch.max(model_output, 1)  # Get the index of the max value (class)

    skin_types = ['Dry', 'Oily', 'Sensitive', 'Combination', 'Normal']
    predicted_skin_type = skin_types[predicted_class.item()]  # Convert to Python scalar
    print(f"\nPredicted Skin Type: {predicted_skin_type}")
    return [predicted_skin_type]
# Recommend Products with Image
def recommend_products_with_image():
    print("📸 Using camera to detect skin type...")

    # Capture skin image
```

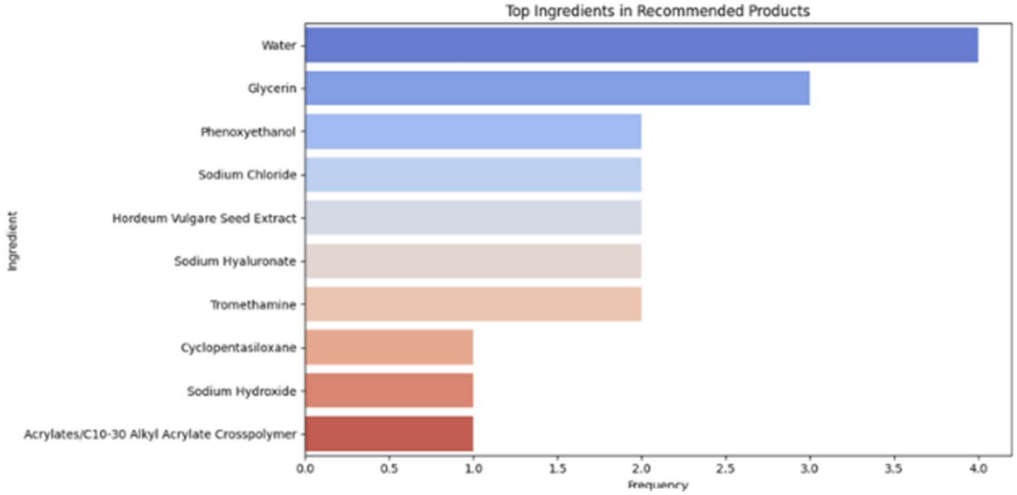
```
    img_path = capture_image()

    # Predict skin type from the image
    skin_types = predict_skin_type(img_path)

    # Recommend products based on detected skin type
    print(f"\nTop product recommendations for: {', '.join(skin_types)} skin 👇 ")
    result = recommend_products(skin_types, top_n=5)
    for r in result:
        print(f"{r[0]} by {r[1]} - Price: ${r[2]} - Rank: {r[3]} - Match Score: {r[4]}")

    ingredient_df = get_ingredient_frequency(result, df)
    plot_ingredient_frequency(ingredient_df)

def main_menu():
    print("✨ Welcome to GlowMatch ✨ ")
    print("How would you like to get skincare recommendations?")
    print("1. Enter skin type manually")
    print("2. Use camera to detect skin type")

    choice = input("Choose option (1 or 2): ")

    if choice == '1':
        recommend_products_interactive()
    elif choice == '2':
        recommend_products_with_image()
    else:
        print("❌ Invalid choice. Please select 1 or 2.")

if __name__ == "__main__":
    main_menu()
```

 # Results

```
✨ Welcome to GlowMatch ✨
How would you like to get skincare recommendations?
1. Enter skin type manually
2. Use camera to detect skin type
Choose option (1 or 2):
```

Option 1:

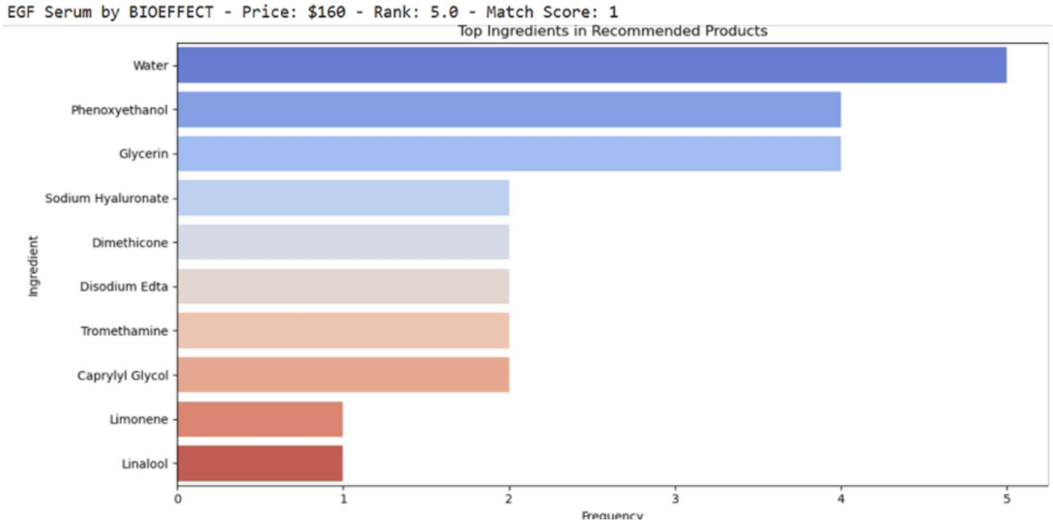
```
✨ Welcome to GlowMatch ✨
How would you like to get skincare recommendations?
1. Enter skin type manually
2. Use camera to detect skin type
Choose option (1 or 2):  1

Please enter your skin type(s) separated by commas.
Options: Dry, Oily, Sensitive, Combination, Normal
Your skin type(s):  dry, sensitive

Top product recommendations for: Dry, Sensitive skin 👇
Pore Refining Detox Double Cleanse by ERNO LASZLO - Price: $55 - Rank: 5.0 - Match Score: 2
Refreshing Gel Cleanser by CLARISONIC - Price: $19 - Rank: 5.0 - Match Score: 2
EGF Serum by BIOEFFECT - Price: $160 - Rank: 5.0 - Match Score: 2
Capture Totale Dreamskin Advanced Refill by DIOR - Price: $128 - Rank: 5.0 - Match Score: 2
30 Day Treatment by BIOEFFECT - Price: $290 - Rank: 5.0 - Match Score: 2
```



Option 2:

```
✨ Welcome to GlowMatch ✨
How would you like to get skincare recommendations?
1. Enter skin type manually
2. Use camera to detect skin type
Choose option (1 or 2):  2
📸 Using camera to detect skin type...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 280ms/step
```


```
Top product recommendations for: Oily skin 👇
Limited Edition Dramatically Different™ Moisturizing Gel by CLINIQUE - Price: $39 - Rank: 5.0 - Match Score: 1
Epidermal Re-Texturizing Micro-Dermabrasion by KIEHL'S SINCE 1851 - Price: $41 - Rank: 5.0 - Match Score: 1
Pore Refining Detox Double Cleanse by ERNO LASZLO - Price: $55 - Rank: 5.0 - Match Score: 1
Refreshing Gel Cleanser by CLARISONIC - Price: $19 - Rank: 5.0 - Match Score: 1
EGF Serum by BIOEFFECT - Price: $160 - Rank: 5.0 - Match Score: 1
```



CONCLUSION

In this project, a hybrid deep learning architecture combining Convolutional Neural Networks (CNN) and Multi-Layer Perceptron (MLP) was implemented to build a personalized skincare recommendation system. The CNN model was responsible for accurately predicting the user's skin type (dry, oily, or normal) from facial images, while the MLP model utilized structured product data to recommend skincare items best suited to the identified skin type.

The CNN model, based on the VGG16 architecture, demonstrated strong performance in classifying skin types by learning visual patterns from preprocessed facial images. It effectively extracted deep image features relevant to skin characteristics, enabling accurate categorization. Evaluation metrics such as accuracy and confusion matrix showed that the model could reliably identify the correct skin type in most cases.

The MLP model, trained on skincare product attributes such as ingredients, price, rating, and skin suitability indicators, played a crucial role in filtering and prioritizing personalized product recommendations. It was further supported by ingredient frequency analysis using the Counter module, allowing users to understand which components were most common across the recommended products. The combination of deep learning and data-driven product analysis resulted in meaningful and context-aware recommendations.

Overall, the dual-model system proved to be an effective solution for automating and personalizing skincare recommendations. By leveraging CNN for visual analysis and MLP for structured decision-making, the project highlights the potential of artificial intelligence in enhancing personal care applications. This approach offers scalability, user engagement, and practical utility, with future enhancements potentially including real-time skin health monitoring, a broader product database, and integration into mobile or e-commerce platforms.

REFERENCES

1. https://pytorch.org/docs/stable/index.html
2. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7775696/
3. https://cs.unc.edu/~weiss/skin/skin.pdf
4. https://incidecoder.com/
5. https://www.sciencedirect.com/science/article/pii/S2405844023083846
6. https://medium.com/@dzakiahptr/skincare-products-recommendation-system-7d3b2c94884c
7. https://www.researchgate.net/publication/378519440_Deep_learning-based_skin_care_product_recommendation_A_focus_on_cosmetic_ingredient_analysis_and_facial_skin_conditions
8. https://ieeexplore.ieee.org/document/10459793