AI based skincare Assistant

***Abstract*—In the 21st century world, where the global community is consuming skincare and beauty products on a daily basis, facial skin can represent various types of body reactions, from blackheads to cancer spots. It is necessary to maintain healthy skin since that plays a crucial role in controlling face condition and safeguarding it. This document suggests an AI-based skincare assistant that analyzes facial issues and creates customized skincare routines using appropriate products based on skin type. This paper used a deep learning algorithm-based facial skin image classification system using Convolutional Neural Networks(CNN) deep learning algorithm, and it can recognize four classes of facial skin problems, acne, blackhead, carcinoma, wrinkles, and normal skin condition. Finally, by understanding and classifying the issue assistant offers skin treatment according to skin type. The main evaluation of the project is done using the data set of facial photos, and findings show the most suitable model and how well it can generate skin care advice and assists users in having better skin.**

***Index Terms*—skincare , CNN, tensorflow, keras, deep learning algorithm**

I. Introduction

Due to the rapid development of the world community and more serious health concern, it is necessary to maintain a healthy face condition nowadays. The whole process of keeping the face clear and fresh might be challenging cause not everyone can define the facial disease and provide sophisticated healing products.

Skin conditions may seem innocuous, but if they are not properly treated, they can be harmful. A noticeable percentage of the population suffers from moderate to severe facial skin degeneration, which can result in further problems such as acne and eczema [1]. Therefore, basic analyzing and understanding of facial conditions are important for health and personal well-being, and this project suggests creating an AI-based skincare analysis and recommendation system that can offer customized recommendations to people based on their skin type and facial issues.

The advancements in deep learning algorithms, particularly in image processing and facial recognition, form the foundation of the suggested skincare assistant. Convolutional neural networks (CNNs) are used by the system to precisely identify different skin conditions in facial photos [2]. Additionally, the system includes a suggestion engine that makes use of skincare conditions to deliver even more individualized recommendations.

This project aims to create an AI-based skincare assistant that can detect skin issues, make individualized suggestions, and help users with their skincare regimens. The research intends to demonstrate how AI can be used to improve skin care procedures and user experiences.

These are the main contributions of this work:

- We present an AI-based skincare assistant that, by utilizing artificial intelligence, outperforms conventional skincare methods. Our skincare assistant, in contrast to traditional methods, makes precise and individualized skincare recommendations based on detected skin type with modern algorithms rather than generic observations of skin ailments and appearance.
- Acne, wrinkles, blackheads, and other sorts of facial skin problems are among the many different facial skin issue types on which the skincare assistant is taught. The assistant can learn and recognize the distinctive characteristics of each type of skin condition, enabling reliable detection and analysis through deep learning techniques and feature extraction.
- Additionally, we present an implementation of an AI-based skincare assistant as a website that is affordable and convenient for everyone.

The summary of related works and their contributions may be found in the subsection. The dataset and all of the preprocessing methods are in Section 2. In Section 3, a full explanation of the model and CNN layers are explained. Implementation of the model in the website is showed Section 4. The investigated results and discussion of the models can be found in Section 5.

A. Related Work

Various scholars have conducted a significant amount to explore the use of deep neural networks, particularly convolutional neural networks (CNNs), for diagnosing facial skin conditions. Kwon and da Vitoria Lobo [3] focused on differentiating between younger and older individuals by utilizing filters to separate wrinkles from facial images and analyzing facial folds and irregularities in different areas. Alarifi et al. [4] classified various types of facial skin issues, including normal skin with spots, patches with roughness, and wrinkles, employing SVM and CNN models.

Maroni et al. [5] developed a prototype framework for automated acne diagnosis, incorporating techniques such as body part identification, skin segmentation, heat mapping, and acne extraction. They utilized the Haar-Cascade classifier and Random Forest algorithms to achieve these tasks. Hameed et al. [6] proposed a hybrid approach combining the Naive Bayes Classifier and image processing techniques for the detection of skin pimples.

In terms of identifying small-sized objects, Gessert [7] proposed a deep neural network system that utilized segmented sections from high-resolution skin photos. This model consisted of convolutional and recurrent layers and was designed to handle imbalanced data. Additionally, Cui [8] introduced a modified Single Shot Detector (SSD) structure with fusion layers and deconvolutional layers for detecting small-size objects.

To solve numerous face skin issues, Bekmirzaev [9] suggested a segmentation model architecture including a combination of LSTM layers and convolutional layers. Furthermore,

Jung et al. [10] developed a DL-based model with high accuracy and a low loss rate to classify and predict four distinct types of facial acne. They collected data from various legally available sources and conducted their Python experiments with Keras, Tensorflow, and Scikit-Learn.

In addition to skin condition diagnosis, researchers have also explored methods for enhancing the quality of input images, particularly by increasing image resolution. Various techniques have been proposed for image enhancement.

While deep learning models have shown exceptional performance in automatically analyzing skin lesions, they often require a large amount of labeled clinical data, which is not readily available. The size of the training data set may be increased by using data augmentation techniques, such as editing the input photos [11].

Despite the significant contributions of previous studies in applying deep learning to diagnose facial skin conditions, they often overlook the technical challenges associated with such diagnoses. In contrast, this study aims to identify specific challenges and provides appropriate recommendations based on the outcomes, accompanied by a detailed explanation of deep learning techniques. Moreover, besides identifying the facial issue, it will provide a unique recommendation engine to solve it.

II. Dataset

Dermnet, the top free dermatology resource in the world, provided the dataset on skin issues used in this study. The dataset was carefully chosen to be trained and tested for skin type. It is made up of a relevant variety of pictures that include appropriate metadata about various skin problems. On top of that, open source was used to compile the product dataset.

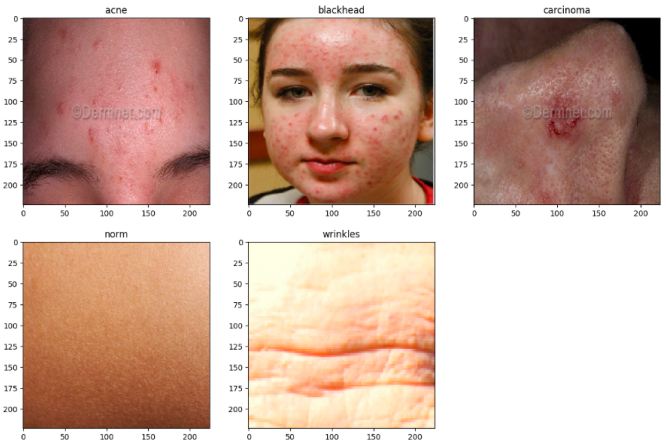

Fig. 1. Used five classes in dataset

A. Data characteristics and attributes

For the purposes of training the skincare assistant and doing skin analysis, the dataset contains a number of crucial qualities and features. High-resolution photos of skin types. The AI model uses these photos as input so that it can analyze them and provide recommendations. The usage of a big data set and well-trained model gives more accurate predictions. Wrinkles, acne, blackheads, and other skin issues on the face are common [1], as seen in Fig. 1. There are five classes of facial skin problems, which are acne, blackhead, wrinkles, carcinoma, and normal skin condition. The completed system can help people to investigate their skin and find solutions to the issue.

The AI model is trained using these labels as the basis for the classification and evaluation of skin diseases. The dataset of products includes metadata for several skincare products, including information on the brands, components, URL, price, rank, and product categories (moisturizer, cleanser, serum, etc.).

	Skin Condition						
	Acne	Black-head	Carci-noma	Wrinkle	Norm.	Total	After Aug.
Test	33	50	31	40	14	115	
Train	131	193	124	158	56	662	1527

Dataset table

B. Data preprocessing

The skincare dataset went through preprocessing techniques to assure data consistency and quality before training the AI model.

- Image processing: An image is resized by the rescaling procedure by a chosen factor. A single floating point value or multiple values, one for each axis, can be used as the scaling factor.
- Label Encoding: Using the proper encoding methods, labels for skin conditions were translated into numerical representations. This made it easier to categorize and analyze skin problems throughout the development and testing of models.
- Data Augmentation: Data augmentation techniques, including rotation, flipping, and zooming, were used in the photos to increase the diversity and robustness of the dataset. This procedure enhanced the model's capacity to generalize by preventing overfitting [11]. Additionally to our training set, 1246 changed images were added.
- Splitting To Testing and Training Data: In order to ensure an appropriate distribution of data for model training and monitoring, the dataset was separated into training(0.80) and testing(0.20) sets. To prevent prejudice during training, the dividing was done while keeping class balance.

III. Methodology

Tensorflow and Keras libraries were used to define the skin type and issue. A free and open-source software library for artificial intelligence and machine learning is called TensorFlow. Although it may be applied to many different tasks, it focuses especially on the training and inference of deep neural networks [12]. Our model was created using Keras primarily because it supports convolutional and recurrent neural networks. Other widely used utility layers are supported, including dropout, batch normalization, and pooling. Team members were skilled.

Because Keras offers categorical_crossentropy, the labels of the dataset deduce the directory's name and are saved as categorical vectors. Since the dataset contains many classes, this label's mode was used. The term "seed" was also employed to prevent data sequence repetition. The seed is a starting value that guarantees that the same arrangement of elements won't be picked. This value will be applied to the library we use to shuffle our data.

Since the dataset was too tiny for deep learning, overfitting was likely to occur. Our initial attempt to train our dataset using Keras' layer augmentation techniques, such as random flip, random rotation, and random zoom, resulted in lengthy training times and overfitting. Results were better when using the Python module Augmentor, which applied augmentation before training. A Python image augmentation package called "!Augmentor" is used for machine learning. It intends to be a standalone library that is not dependent on platforms and frameworks, making it more practical, enabling finer-grained control over augmentation, and implementing the augmentation methods that are most applicable in real-world settings. It uses a stochastic method that assembles operations into a pipeline by using building pieces [13].

Images were randomly flipped from left to right with a frequency of 50% and from top to bottom with a probability of 30%. Our images are rotated between 10 and -10 degrees with a 70% chance of being taken. The rotational formula for a given angle (theta) is shown below.

$$x = x \cdot \cos(\theta) - y \cdot \sin(\theta)$$
$$y = x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

$$\begin{bmatrix} A & B & C \\ D & E & F \end{bmatrix} => \begin{bmatrix} F & B & A \\ C & E & D \end{bmatrix}$$

The Augmentor library utilized the pillow library [13] to read the image. PIL's mode represents the type and depth of each pixel in an image and is specified as a string [14]. The mode "RGB" indicates a true-color image with 3x8-bit pixels. Each pixel takes advantage of the entire bit depth, ranging from 0 to 1 for a 1-bit pixel, 0 to 255 for an 8-bit pixel, INT32 for a 32-bit signed integer pixel, and FLOAT32 for a 32-bit floating-point pixel. PIL follows the ITU-R norm for picture encoding and signal properties of high-definition television (HDTV), converting an RGB image linearly adjusted according to ITU-R 709 and using the D65 luminant to the CIE XYZ color space [15].

$$\begin{bmatrix} \frac{R_{1,1}}{255} & \frac{G_{1,1}}{255} & \frac{B_{1,1}}{255} \\ \frac{R_{1,2}}{255} & \frac{G_{1,2}}{255} & \frac{B_{1,2}}{255} \\ \frac{R_{2,1}}{255} & \frac{G_{2,1}}{255} & \frac{B_{2,1}}{255} \\ \frac{R_{2,2}}{255} & \frac{G_{2,2}}{255} & \frac{B_{2,2}}{255} \end{bmatrix}$$

The normalization step plays a vital role in image preprocessing. The rescale procedure is employed to resize the image by a specified factor. The scaling factor can be a single floating-point number or multiple values, with each value corresponding to a specific axis. Most digital pictures have pixels with intensities in the 0 to 255 spectrum, with 0 denoting black and 255 denoting white. To normalize the pixel values, the following formula is applied:

$$x = x \cdot \frac{1}{255.0}$$

TensorFlow offers several features to enhance the efficiency and speed of training models. The cache() function saves data, resulting in faster calculations. The shuffle operation randomly reorders the components of the dataset. The standard approach is to generate a new set of data that has been shuffled by picking random samples from a buffer that has been pre-sized. Shuffling introduces randomness into the training process and prevents the model from recognizing patterns or orders in the input.

A. CNN architecture

The Convolutional 2D Layer plays a critical role by applying a convolutional kernel to the input data. This kernel, characterized by an integer or a pair of values, determines the dimensions of the 2D convolution window. Our model employs convolutional layers with 32, 64, and 128 filters, respectively. Each filter detects specific features or characteristics within the input image [16]. During the convolutional operation, the kernel is convolved with the input image. This process entails sliding the kernel across the image and performing a dot product between the corresponding pixel values in the input and the kernel weights. This computation is performed at every location in the image, resulting in a tensor of output values representing the filters' activations or responses.

The primary distinction between 2D and 3D convolutions lies in the input and kernel dimensionality. In 2D convolutions, the input is typically a 2D image with height and width, while the kernel is a 2D matrix. The convolution operation is performed spatially by scanning the kernel across the image. On the other hand, 3D convolutions are employed when the input data has an additional dimension, such as depth or time[17]. In this case, the input and kernel are 3D tensors, and the convolution is performed across all three dimensions. This enables the model to capture spatial and temporal features in volumetric or sequential data.

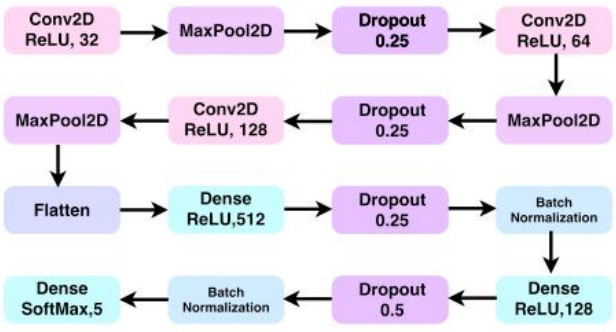

Fig. 2. CNN - Architecture

Max Pooling 2D: The purpose of the Max Pooling 2D operation is to downsample the input data along its spatial dimensions, namely height and width. This downsampling is achieved by taking the maximum value within an input window of a specific size, determined by the pool_size parameter. During the pooling operation, a window of specified dimensions is moved across the input data, and at each position, the maximum value within the window is selected. This process is performed independently for each input channel. In our case, the pooling window size is defined as $(2, 2)$, indicating a 2×2 window.

The maximum value within each 2×2 pooling window is recorded.

By applying the Max Pooling 2D operation, the spatial dimensions of the input data are effectively reduced, resulting in a downsampled output. The output layer's spatial shape is determined by factors such as the size of the input, the pooling window size, and the strides used in the pooling operation.

$$OutputLayerSize = \frac{(InputLayerSize - PoolSize)}{Stride} + 1$$

Max pooling reduces the dimensionality of images by reducing the number of pixels in the output from the prior convolutional layer.

$$\mathrm{R} = \begin{bmatrix} 0.267 & 0.376 & 0.489 & 0.178 \\ 0.244 & 0.457 & 0.612 & 0.532 \\ 0.421 & 0.533 & 0.234 & 0.112 \\ 0.462 & 0.821 & 0.598 & 0.334 \end{bmatrix}$$

$$\mathrm{G} = \begin{bmatrix} 0.344 & 0.216 & 0.542 & 0.199 \\ 0.245 & 0.512 & 0.342 & 0.223 \\ 0.466 & 0.555 & 0.423 & 0.123 \\ 0.789 & 0.642 & 0.267 & 0.389 \end{bmatrix}$$

$$\mathrm{B} = \begin{bmatrix} 0.127 & 0.109 & 0.546 & 0.276 \\ 0.427 & 0.279 & 0.186 & 0.377 \\ 0.571 & 0.217 & 0.323 & 0.225 \\ 0.288 & 0.235 & 0.521 & 0.491 \end{bmatrix}$$

Algorithm 1 Computing output feature map

Input : R, G, B

Output: output_feature_map[0, 0]

$output\_feature\_map[0, 0] \leftarrow [\max(R[0:2, 0:2]),$

$\max(G[0:2, 0:2]), \max(B[0:2, 0:2])$

return $output\_feature\_map[0, 0]$

Dropout Regularization: Dropout regularization is frequently used to improve the results of deep neural networks and reduce overfitting. A certain number of layer outputs are "dropped out" or disregarded at random during training. This has the result of having the layer appear to have a different number of nodes and connections to the preceding layer and be regarded as such. During training, a layer is updated with a different "view" of the previous layer.

To temporarily remove a unit from the network and all of its incoming and outgoing connections is to drop it out [**?**]. It implies that 25% of the input units will have their values adjusted randomly to 0 throughout each training stage. To account for the units that were dropped, the units that are not set to 0 are scaled by a factor of :

$$\frac{1}{1 - dropout\_rate}$$

The formula for scaling the non-dropped units is :

$$(1 - dropout\_rate) \cdot x \cdot 1$$

For each training phase, 25% of the 32 input feature map channels will have their values randomly adjusted to 0. To make up for the dropped units, the non-dropped units will be scaled by a factor of $\frac{4}{3}$.

Flatten: The flattening layer creates a one-dimensional array from our input map. All of the components along the spatial dimensions (Height and Width) are concatenated. Following complete convolutional processing, we combine the layers into a single array and send it to the dense layer.

Dense layers have strong connections to their previous layers, meaning that every neuron in the layer is also linked to every neuron in the layer above it. Each neuron in the lower layer communicates with the neurons in the dense layer, which multiply matrices and vectors in a model. During matrix-vector multiplication, the row vector of the output from the preceding layers is the same as the column vector of the dense layer. The row vector must have the same number of columns as the column vector in order to multiply matrices with vectors.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{1}$$

X is a matrix with a diagonal of 1, and A is a (M x N) matrix. The values inside the matrix are the learned parameters of the earlier layers, and backpropagation may likewise be used to update them. The most popular algorithm for training feedforward neural networks is backpropagation. Backpropagation generally determines the gradient of the loss function in a neural network with respect to the network weights for a single input or output. According to the initial idea, the dense layer will produce an N-dimensional vector as its output. We can observe that the vectors' dimension is being reduced. As a result, each neuron in a dense layer is employed to change the vectors' dimension.

Backpropagation Algorithm:

1. Initialize the weights and biases randomly.

2. Forward Propagation: Compute the predicted output $\hat{y}$ for a given input x using the current weights and biases:

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}, \quad \text{(weighted sum)}$$
$$a^{[l]} = g^{[l]}(z^{[l]}), \quad \text{(activation function)}$$

3. Compute the cost function J: Compare the predicted output $\hat{y}$ with the true output y and compute the cost:

$$J = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right),$$

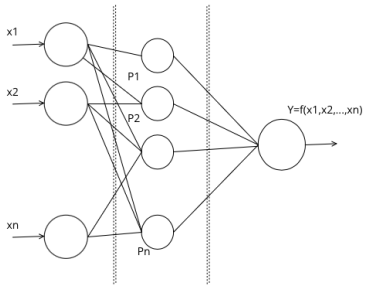

Fig. 3. Backpropagation Algorithm

where log represents the natural logarithm.

4. Backward Propagation: Compute the gradients of the weights and biases with respect to the cost function:

$$dz^{[l]} = \frac{\partial J}{\partial z^{[l]}} = a^{[l]} - y,$$
$$dW^{[l]} = \frac{\partial J}{\partial W^{[l]}} = \frac{1}{m} dz^{[l]} a^{[l-1]T},$$
$$db^{[l]} = \frac{\partial J}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dz^{[l](i)},$$
$$da^{[l-1]} = \frac{\partial J}{\partial a^{[l-1]}} = W^{[l]T} dz^{[l]},$$

where m is the number of training examples.

5. Update the weights and biases: Adjust the weights and biases using the gradients and a learning rate α:

$$W^{[l]} := W^{[l]} - \alpha dW^{[l]},$$
$$b^{[l]} := b^{[l]} - \alpha db^{[l]}.$$

6. Repeat steps 2-5 until convergence or a desired number of iterations is reached.

IV. Software

Our model comprises two dense layers with 512 and 128 ReLU neurons. It was decided to chose five since there are five classes, and we used softmax to determine the likelihood that an image belongs to that class.

A vector of values is transformed into a probability distribution via Softmax. The output vector's members are in the range (0, 1) and add up to 1. Each vector is treated separately. The axis parameter determines the axis of the input the function is applied along. The Softmax activation is typically used as the final layer of a classification network since the result may be interpreted as a probability distribution. Each vector's softmax is calculated as :

$$softmax(x) = \frac{\exp(x)}{\sum_{i=1}^{N} \exp(x_i)}$$

ADAM: A new development in deep learning algorithms for computer vision and natural language processing is the Adam optimization technique, a stochastic gradient descent extension. In place of the conventional stochastic gradient descent method, Adam is an optimization technique that may be used to iteratively update network weights depending on training data[18]. For all weight updates, stochastic gradient descent maintains a constant learning rate (referred to as alpha), which does not fluctuate throughout training. Alpha = lr = 0.001, beta = 0.9, beta 2 = 0.999, epsilon = 1e-08, decay = 0.0.

Also known as the step size or learning rate. The frequency of updating weights (for example, 0.001). Prior to the rate being modified, initial learning proceeds more quickly for larger numbers (such as 0.3). Smaller values (such as 1.0E-5) drastically reduce learning during training. beta1. the initial instant estimations' exponential decline rate (for example, 0.9). beta2. The rate of exponential decay for second-moment estimations, such as 0.999. In situations with a sparse gradient (like NLP and computer vision issues), this value should be set close to 1.0.

A very small number, such as $(e)^{-8}$, is often used to prevent any division by zero in the implementation.

The decay rate, denoted as α, is updated according to the formula

$$\alpha = \frac{\alpha}{\sqrt{t}}$$

where t represents the epoch.

When a model is being trained, it produces predictions based on a batch of input data, calculates the loss (a measure of the difference between the predictions and the real values), and then computes the gradients of the loss with respect to the parameters. These gradients show how altering the settings might affect the loss. These gradients are used by the optimizer, like Adam, to iteratively update the model's parameters. The optimizer attempts to minimize the loss function and enhance the performance of the model by shifting the parameters in the opposite direction as the gradient.

In machine learning, the gradient, which stands for the slope of change of a function, is used to optimize the model's parameters by shifting them in a way that reduces the loss function. Using a gradient for backpropagation.

Loss: A categorical_crossentropy we use it because we have multiple classes the main idea is measuring the discrepancy between predicted class probabilities and true class labels.

A. Web Application

To implement the model to the HTML web page, we used the Flask environment. Installed Flask and created the required folders and files to set up a Flask project[19]. This normally comprises installing Flask, setting up a virtual environment, and developing the primary Flask application file by using a library like TensorFlow to import the model's h5 file. The model may be loaded using the `load_model` function when the necessary libraries have been imported. While implementing, our model needs to be loaded by TensorFlow, converted to an array, and rescaled object to make predictions. We use the model predict function. It will apply the same layers from our model and give us the output. After that, we use argmax to get the maximum probability and find which problem we have.

Model Implementation

To address the problem statement outlined in Section 2, we designed and implemented a deep learning model using the TensorFlow framework. The model aims to classify skin images into one of five categories: acne, blackhead, carcinoma, normal, and wrinkles. We developed a web-based interface using the Flask framework to allow users to interact with the trained model. Installed Flask and created the required folders and files to set up a Flask project. This normally comprises installing Flask, setting up a virtual environment, and developing the primary Flask application file by using a library like TensorFlow to import the model's h5 file. The model may be loaded using the `load_model` function when the necessary libraries have been imported. While implementing, our model needs to be loaded by TensorFlow, converted to an array, and rescaled object to make predictions. We use the model predict function. It will apply the same layers from our model and give us the output. After that, we use argmax to get the maximum probability and find which problem we have.

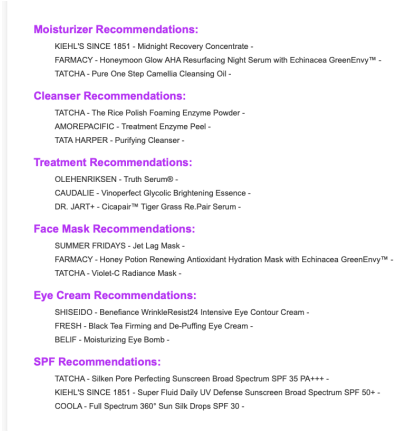

Fig. 4. Web - page with implemented Model

The web application enabled users to upload skin images and receive real-time predictions regarding the skin condition and recommended treatment.

While implementing our model problem occurred with processing. Our model takes only parts of the face and not the whole face. Through implementing pre-trained models: the facial landmark model `shape_predictor_68_face_landmarks` and the One Eye Haarcascade model. `shape_predictor_68_face_landmarks` defines the shape of a face and its parts as long as two eyes are open. One Eye model is used if the shape predictor can't find both eyes. Shape predictor's main points:

After determining the skin patches program will crop and save them in the directory. The forehead, right cheek, left cheek, chin, and nose are skin patches. All of them will be going in the loop to predict skin problems. All predictions will be stored in an array. In this array through formulas :

Skin type will be detected. Seeing skin type user can enter budget and rating. Based on this information site will provide recommended products like cleansers, toners, serums, creams, and eye creams.

Algorithm 2 Processing Facial Landmarks

$face\_x\_min \leftarrow \max(0, \min(\text{landmarks}_{:,0}))$
$face\_x\_max \leftarrow \min(\text{img\_width}, \max(\text{landmarks}_{:,0}))$
$face\_y\_min \leftarrow \max(0, \min(\text{landmarks}_{:,1}))$
$face\_y\_max \leftarrow \min(\text{img\_height}, \max(\text{landmarks}_{:,1}))$
$face\_height \leftarrow face\_y\_max - face\_y\_min$
$forehead\_height \leftarrow$ face_height $\times$ forehead_ratio
$new\_face\_y\_min \leftarrow \max(0,$ face_y_min $-$ forehead_height$)$
$right\_brow\_landmarks \leftarrow \text{landmarks}_{\text{RIGHT\_BROW\_POINTS},:}$
$left\_brow\_landmarks \leftarrow \text{landmarks}_{\text{LEFT\_BROW\_POINTS},:}$
$right\_eye\_landmarks \leftarrow \text{landmarks}_{\text{RIGHT\_EYE\_POINTS},:}$
$left\_eye\_landmarks \leftarrow \text{landmarks}_{\text{LEFT\_EYE\_POINTS},:}$
$mouth\_landmarks \leftarrow \text{landmarks}_{\text{MOUTH\_POINTS},:}$ =0

Algorithm 3 Computing skin type

if *acne_percentage* $>$ 0.4 **then**-
0:
set skin type as sensitive
wrinkles_percentage > 0.2
set skin type as dry
blackhead_percentage > 0.3
set skin type as oily
normal_percentage > 0.2
set skin type as normal **else**
0:
set skin type as combination
0:
return the skin type =0

The deployed model ensured accessibility and usability for individuals seeking information about their skin health. The implementation of our skin image classification model demonstrated its ability to accurately classify skin images into different categories, providing valuable insights for dermatological diagnosis and treatment. The trained model's performance was evaluated rigorously.

V. Results

Below graph has different models' accuracy. We wanted to test which one is better on the early stages of our project. Augmentation was used as layer.

Model	Optimizer	Conv2D Filters	Other Details
Model, 1	Adam	16, 32, 64	Dropout: 0.2
Model, 2	N/A	32, 64, 128	Dropout: 0.4
Model, 3	SGD	N/A	Epochs: 10 Learning Rate: 0.001 Decay: lrate/epochs Momentum: 0.9 Nesterov: False Dropout: 0.2

Model Details

The momentum term is specified by Momentum, which makes it easier for the optimizer to go across the parameter space.

$v = \text{momentum} + \text{velocity} - \text{lr} \cdot \text{gr}$

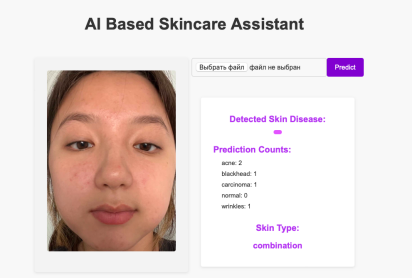

Fig. 5. Web - page with implemented Model

Update parameters in SGD as : $p = p + v$

It is customary to use a value of 0.9, which denotes that the optimizer updates the parameters by accounting for 90% of the prior gradient direction. Decay returns the learning rate decay to the predetermined amount. As training advances, it gradually lowers the learning rate over each epoch, enabling the model to make smaller updates. Model 1 uses the Adam optimizer, which is known for its adaptive learning rate and efficient handling of sparse gradients. The Momentum and Decay terms specified in Model 3 indicate the use of the SGD optimizer with momentum-based updates and a learning rate decay. However, in this case, the Adam optimizer used in Model 1 is considered superior for the given scenario. The SGD and larger hyperparameters weren't helpful due to the limited size of our dataset.Our accuracy increased after rescaling and augmentation were done without using layers.

Model 1 has a smaller architecture with convolutional layers having 16, 32, and 64 filters. This architecture may be more suitable for the limited size of the dataset, preventing overfitting and capturing important features effectively. On the other hand, Model 2 has larger filters, and Model 3 does not provide details about the number of filters, making Model 1's architecture more appropriate for the given data constraints. The mention of rescaling and augmentation being done without using layers in Model 1 suggests that the data was preprocessed separately before training the model. This step could have contributed to enhancing the performance of Model 1, as appropriate data scaling and augmentation can improve the model's ability to learn meaningful patterns.

Considering all these factors, it appears that Model 1, with its smaller architecture, Adam optimizer, appropriate dropout rate, and carefully applied scaling and augmentation techniques, was the most suitable and successful in this scenario. The SGD optimizer with momentum and larger hyperparameters in Model 3, as well as the more complex augmentation techniques in Model 2, may not have been as beneficial due to the limited size of the dataset.

The product recommendation engine was successful because there was used a simple algorithm that finds in the CSV of the skincare products the appropriate item Fig.4 according to the identified skin type and issue and other user-based features such as price and rank.

Challanges

The project work faced several challenges, including:

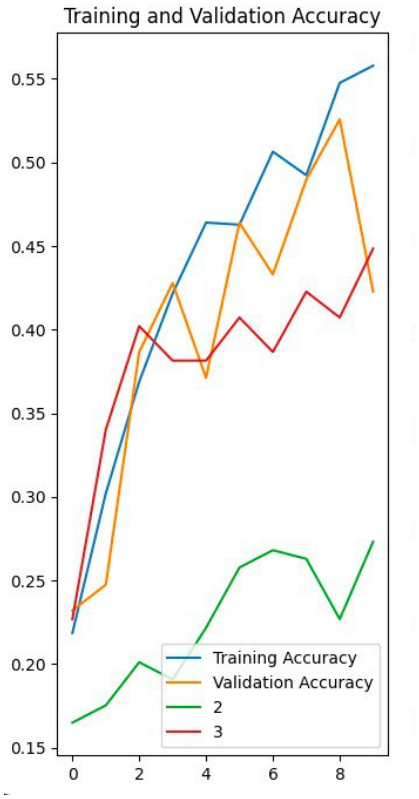

Fig. 6. Difference in Accuracy

Small Dataset: One of the major challenges was working with a small dataset. Limited data can make it difficult to train accurate and robust models, as they may not capture the full complexity and diversity of the problem domain.

Unaffordable Dataset: The majority of the dataset being unaffordable posed another challenge. Acquiring and labeling large datasets can be costly, restricting access to a broader range of samples and potentially limiting the model's ability to generalize well to unseen data. Limited Image Processing: The dataset being suitable only for small image processing tasks presented constraints. The presence of only piles and other issues in the images required additional preprocessing steps, such as cropping the face image, to extract the relevant features for running the model. This extra processing can introduce additional complexity and potential errors.

Overfitting: Overfitting is a common challenge in machine learning projects, and it can arise when models become too complex or when the dataset is insufficient. Overfitting occurs when a model learns to perform well on the training data but fails to generalize to new, unseen data. Dealing with overfitting requires careful regularization techniques and model evaluation to ensure optimal performance.

Implementation in JSON: Initially, the model was implemented in JSON format. However, this approach was not successful due to conflicts between TensorFlow.js and Keras. This compatibility issue between the frameworks may have caused difficulties in correctly representing and training the model.

These challenges highlight the importance of dataset size,

quality, preprocessing steps, regularization techniques, and framework compatibility in machine learning projects. Overcoming these challenges often requires careful consideration, experimentation, and adaptation to ensure the success of the project.

CONCLUSION

The study's conclusion provides a CNN-based skincare assistant model that demonstrates accurate facial acne picture detection and classification with a small amount of data.

Despite all obstacles, the AI-based skincare assistant has demonstrated the potential to offer tailored skincare advice. The assistant may aid customers in understanding their unique skincare requirements and recommend customized regimens to address such problems by utilizing its CNN-based facial issue identification technology. This not only improves user experience but also helps to spread the word about better skincare practices.

Additionally, the use of an AI-based technique enables scalability and adaptation when new skincare issues and products appear. To increase its accuracy and provide more exact skincare advice, the system may be regularly updated and educated on fresh data.

It is crucial to remember that the quality and variety of the dataset, as well as constant optimization and refining of the underlying algorithms, are necessary for the skincare assistant to be successful. To guarantee that the assistant keeps improving and continues to offer precise and trustworthy skincare suggestions, frequent updates and user input are essential. It has a big chance to be used everywhere and be affordable to everyone.

Overall, the AI-based skincare assistant is a big step towards individualized skincare solutions, giving people the power to choose their skincare regimens wisely and encouraging good skin habits. This technology has the potential to change the skincare market and enhance customers' general well-being with more research and improvement.

REFERENCES

[1] Ramli, R., Malik, A. S., Hani, A. F. M., Jamil, A. (2012). Acne analysis, grading and computational assessment methods: an overview. Skin research and technology, 18(1), 1-14.

[2] Chua, L. O., Roska, T. (1993). The CNN paradigm. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 40(3), 147–156.

[3] Kwon, Y. H., da Vitoria Lobo, N. (1999). Age classification from facial images. Computer Vision and Image Understanding, 74(1), 1–21.

[4] Alarifi, J.S., Goyal, M., Davison, A.K., Dancey, D., Khan, R., Yap, M.H. (2017). Facial Skin Classification Using Convolutional Neural Networks. In: Karray, F., Campilho, A., Cheriet, F. (eds) Image Analysis and Recognition. ICIAR 2017. Lecture Notes in Computer Science(), vol 10317. Springer, Cham.

[5] G. Maroni, M. Ermidoro, F. Previdi and G. Bigini, "Automated detection, extraction and counting of acne lesions for automatic evaluation and tracking of acne severity," 2017 IEEE Symposium Series on Computational Intelligence (SSCI), Honolulu, HI, USA, 2017, pp. 1-6.

[6] Hameed, Abdul-Adheem Awad, Waleed Irsan, Nawar Shawkat, Azmi Abdulbaqi,. (2020). Hybrid Technique For Skin Pimples Image Detection and Classification. International Journal of Hydrology Science and Technology. Vol. 29,. 4102-4109.

[7] Gessert, N., Sentker, T., Madesta, F., Schmitz, R., Kniep, H., Baltruschat, I., ... Schlaefer, A. (2019). Skin lesion classification using CNNs with patch-based attention and diagnosis-guided loss weighting. IEEE Transactions on Biomedical Engineering, 67(2), 495-503.

[8] Cui, L., Ma, R., Lv, P. (2020). MDSSD: multi-scale deconvolutional single shot detector for small objects [J]. ece China. Information ences, 63(2).

[9] Bekmirzaev, S., Oh, S., Yo, S. (2019). RethNet: Object-by-Object Learning for Detecting Facial Skin Problems. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (pp. 0-0).

[10] Jung, C., Yeo, I., Jung, H. (2019). Classification model of facial acne using deep learning. Journal of the Korea Institute of Information and Communication Engineering, 23(4), 382–386.

[11] Perez, F., Vasconcelos, C., Avila, S., Valle, E. (2018). Data augmentation for skin lesion analysis. In OR 2.0 Context-Aware Operating Theaters, Computer Assisted Robotic Endoscopy, Clinical Image-Based Procedures, and Skin Image Analysis: First International Workshop, OR 2.0 2018, 5th International Workshop, CARE 2018, 7th International Workshop, CLIP 2018, Third International Workshop, ISIC 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16 and 20, 2018, Proceedings 5 (pp. 303-311). Springer International Publishing.

[12] Joseph, F.J.J., Nonsiri, S., Monsakul, A. (2021). Correction to: Keras and TensorFlow: A Hands-On Experience. In: Prakash, K.B., Kannan, R., Alexander, S., Kanagachidambaresan, G.R. (eds) Advanced Deep Learning for Engineers and Scientists. EAI/Springer Innovations in Communication and Computing. Springer, Cham.

[13] Bloice, M. D., Stocker, C., Holzinger, A. (2017). Augmentor: an image augmentation library for machine learning.

[14] Ye, R. Z., Noll, C., Richard, G., Lepage, M., Turcotte, É. E., Carpentier, A. C. (2022). DeepImageTranslator: A free, user-friendly graphical interface for image translation using deep-learning and its applications in 3D CT image analysis. SLAS technology, 27(1), 76-84.

[15] Umesh, P. (2012). Image processing in python. CSI Communications, 23(2).

[16] Lee, H., Song, J. (2019). Introduction to convolutional neural network using Keras; an understanding from a statistician. Communications for Statistical Applications and Methods, 26(6), 591-610.

[17] Patravali, J., Jain, S., Chilamkurthy, S. (2018). 2D-3D fully convolutional neural networks for cardiac MR segmentation. In Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges: 8th International Workshop, STACOM 2017, Held in Conjunction with MICCAI 2017, Quebec City, Canada, September 10-14, 2017, Revised Selected Papers 8 (pp. 130-139). Springer International Publishing.

[18] Zhang, Z. (2018, June). Improved adam optimizer for deep neural networks. In 2018 IEEE/ACM 26th international symposium on quality of service (IWQoS) (pp. 1-2). Ieee.

[19] Grinberg, M. (2018). Flask web development: developing web applications with python. " O'Reilly Media, Inc.".