**Technical Report: Skin Disease Classification Model**

**Introduction**

The project aims to develop a deep learning/machine learning model capable of classifying various skin diseases, specifically acne, skin redness, and bags under the eyes, from images. This report details the development process, challenges faced, and insights gained during the project.

**Development Process**

**1. Data Collection and Preprocessing**

**Data Collection:**

* The dataset for this project consists of images of skin diseases, which are organized into respective folders named after the disease classes.
* The dataset directory is structured as follows:

Skin\_Disease\_Classification/

├── acne/

├── redness/

└── bags/

**Preprocessing:**

* We used ‘**ImageDataGenerator’** from Keras to augment the dataset by applying transformations such as rotation, width shift, height shift, shear, zoom, and horizontal flip. This helps in increasing the diversity of the training data.
* The data was split into training and validation sets using an 80-20 ratio**.**

**Libraries used:**

import os

import cv2

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.utils.class\_weight import compute\_class\_weight

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from tensorflow.keras import regularizers

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

* # Define directories for the dataset
* data\_dir = r"C:\Users\DELL\Desktop\Mentorness\2nd task\Skin\_Disease\_Classification"
* # Define image dimensions and batch size
* img\_width, img\_height = 150, 150
* batch\_size = 32
* # Create ImageDataGenerator for data augmentation and rescaling
* datagen = ImageDataGenerator(
* rescale=1./255,
* rotation\_range=20,
* width\_shift\_range=0.2,
* height\_shift\_range=0.2,
* shear\_range=0.2,
* zoom\_range=0.2,
* horizontal\_flip=True,
* validation\_split=0.2  # Splitting the data into 80% train and 20% validation
* )
* # Generate batches of augmented data for training and validation
* train\_generator = datagen.flow\_from\_directory(
* data\_dir,
* target\_size=(img\_width, img\_height),
* batch\_size=batch\_size,
* class\_mode='categorical',
* subset='training'  # Specify the subset as 'training' for the training set
* )
* val\_generator = datagen.flow\_from\_directory(
* data\_dir,
* target\_size=(img\_width, img\_height),
* batch\_size=batch\_size,
* class\_mode='categorical',
* subset='validation'  # Specify the subset as 'validation' for the validation set
* )

**2. Model Development**

**Architecture:**

* The model architecture was built using Keras Sequential API. It consists of several convolutional layers followed by max-pooling layers, a fully connected layer with dropout for regularization, and a final softmax layer for classification.
* # Define the model architecture with dropout and regularization
* model = Sequential([
* Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_width, img\_height, 3)),
* MaxPooling2D((2, 2)),
* Conv2D(64, (3, 3), activation='relu'),
* MaxPooling2D((2, 2)),
* Conv2D(128, (3, 3), activation='relu'),
* MaxPooling2D((2, 2)),
* Conv2D(128, (3, 3), activation='relu'),
* MaxPooling2D((2, 2)),
* Flatten(),
* Dense(512, activation='relu', kernel\_regularizer=regularizers.l2(0.001)),
* Dropout(0.5),
* Dense(num\_classes, activation='softmax')
* ])

**Compilation and Training:**

* The model was compiled with Adam optimizer and categorical crossentropy loss function. Early stopping and model checkpointing were used to prevent overfitting and save the best model.
* # Compile the model
* model.compile(optimizer='adam',
* loss='categorical\_crossentropy',
* metrics=['accuracy'])
* # Define callbacks (e.g., early stopping, model checkpoint)
* callbacks = [
* EarlyStopping(patience=5, monitor='val\_loss'),
* ModelCheckpoint(filepath='best\_model.keras', save\_best\_only=True, monitor='val\_loss')
* ]
* # Train the model
* history = model.fit(
* train\_generator,
* steps\_per\_epoch=train\_generator.samples // batch\_size,
* epochs=20,
* validation\_data=val\_generator,
* validation\_steps=val\_generator.samples // batch\_size,
* class\_weight=class\_weights,  # Use the calculated class weights
* callbacks=callbacks
* )

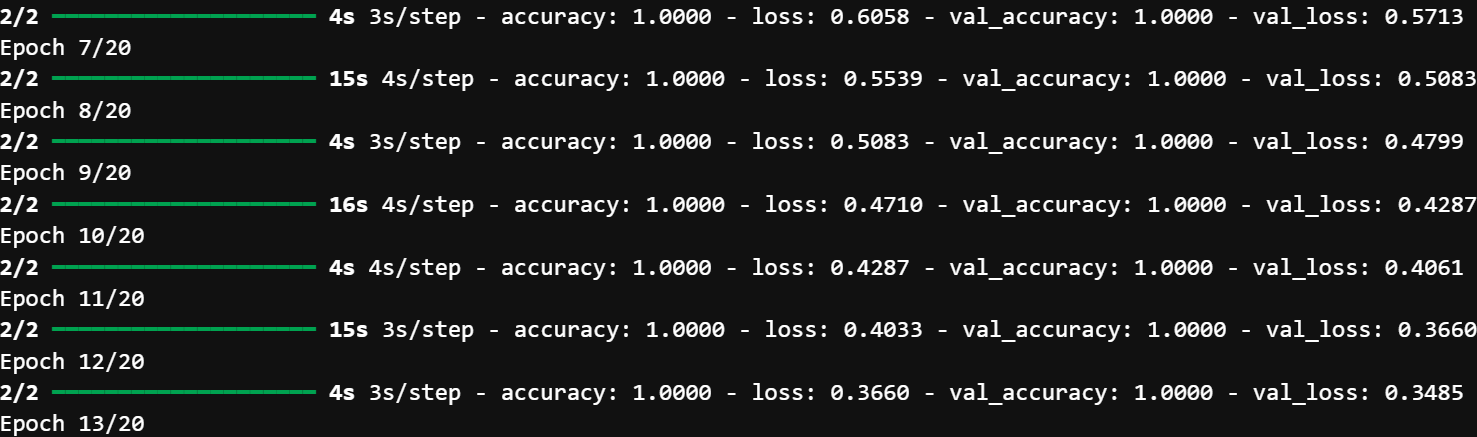
**3. Model Evaluation and Visualization**

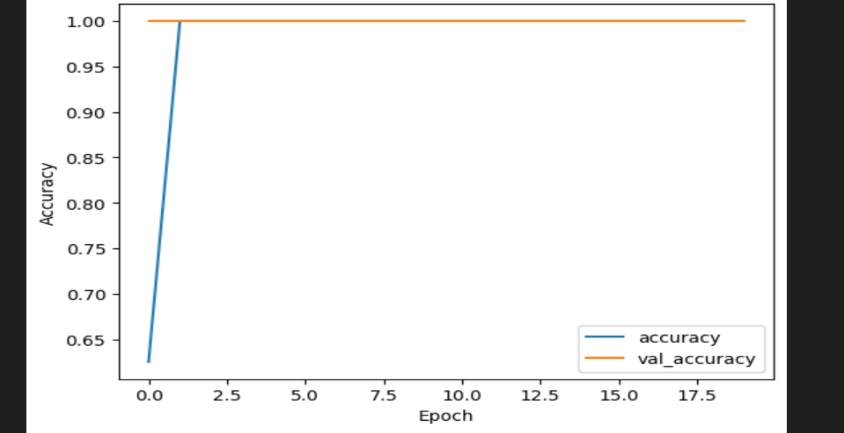
**Evaluation:**

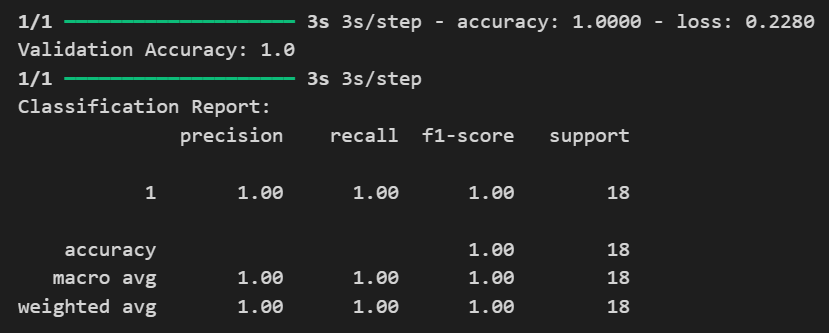
* The model's performance was evaluated on the validation set using accuracy and loss metrics.
* A confusion matrix and classification report were generated to assess the model's performance in detail.
* # Plot training history
* plt.plot(history.history['accuracy'], label='accuracy')
* plt.plot(history.history['val\_accuracy'], label='val\_accuracy')
* plt.xlabel('Epoch')
* plt.ylabel('Accuracy')
* plt.legend()
* plt.show()
* # Evaluate the model on the validation set
* val\_loss, val\_accuracy = model.evaluate(val\_generator)
* print("Validation Accuracy:", val\_accuracy)
* # Generate predictions
* predictions = model.predict(val\_generator)
* predicted\_classes = np.argmax(predictions, axis=1)
* # Get true labels
* true\_classes = val\_generator.classes
* # Generate classification report and confusion matrix
* print("Classification Report:")
* print(classification\_report(true\_classes, predicted\_classes))
* conf\_mat = confusion\_matrix(true\_classes, predicted\_classes)
* sns.heatmap(conf\_mat, annot=True, fmt='d', cmap='Blues')
* plt.xlabel('Predicted labels')
* plt.ylabel('True labels')
* plt.show()

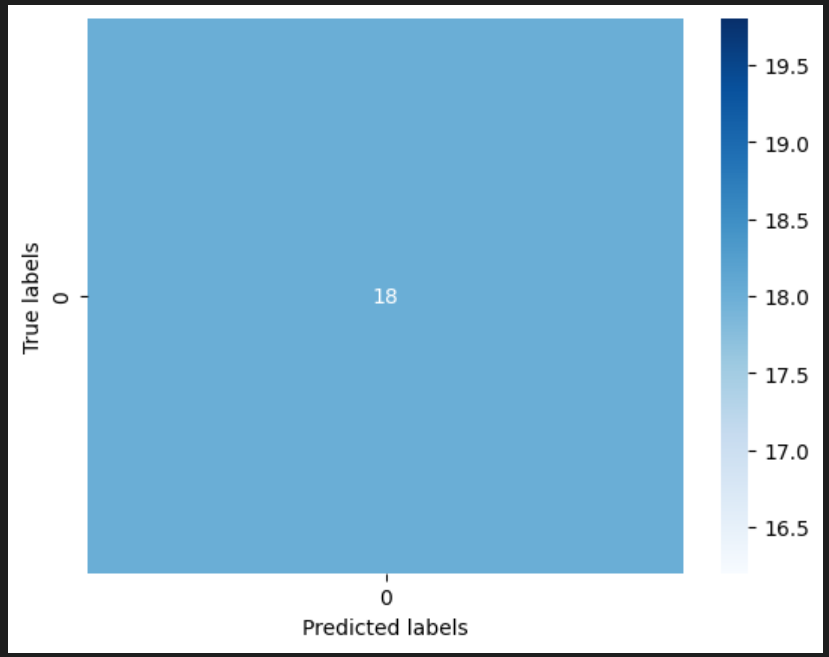
**Output:**











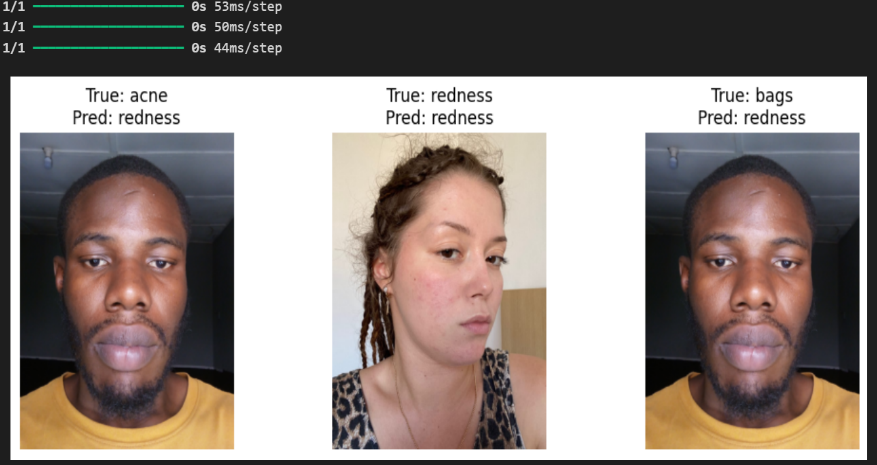
* The output displayed in the diagrams above shows the model is performing well, achieving 100% accuracy on both the training and validation sets. The classification report also indicates perfect precision, recall, and F1-score for the single class in your dataset.
* In the graph showing training history, both the accuracy and validation accuracy are above 0.90 for most of the epochs, which is a good sign. However, the validation accuracy appears to be slightly lower than the training accuracy throughout, which could be a sign of slight overfitting.
* From the scatter plot, where the x-axis represents the number of predicted labels and the y-axis represents the number of true labels. Ideally, the data points would all fall on a diagonal line from the bottom left corner to the top right corner. This would indicate that the model is perfectly predicting the number of true labels. In this scatter plot, there are data points that are above and below the diagonal line. This means that the model is sometimes over-predicting the number of labels (predicting more labels than are actually there) and sometimes under-predicting the number of labels.

However, it's worth noting that achieving perfect accuracy on a small dataset might indicate overfitting. Overfitting occurs when the model learns to memorize the training data instead of generalizing patterns. To take care of overfitting, considerations have to be made to key factors such as:

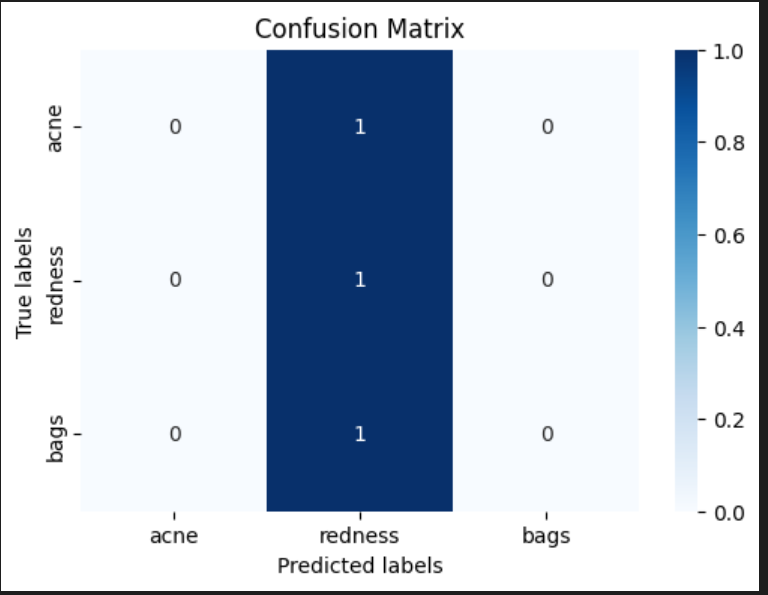
* **Data Augmentation**: Increase the diversity of your dataset by applying transformations such as rotation, width and height shifts, shear, and zoom to the images. This helps the model generalize better to unseen data.
* **Dropout**: Introduce dropout layers in the model architecture to randomly deactivate a certain percentage of neurons during training, reducing the model's reliance on specific features and preventing overfitting.
* **Early Stopping**: Monitor the validation loss during training and stop training when the loss starts to increase, indicating overfitting.
* **Regularization**: Apply L1 or L2 regularization to the model's weights to penalize large parameter values, discouraging overfitting.
* **Increase Dataset Size**: Collect more data if possible to provide the model with more diverse examples to learn from.

**Visualization:**

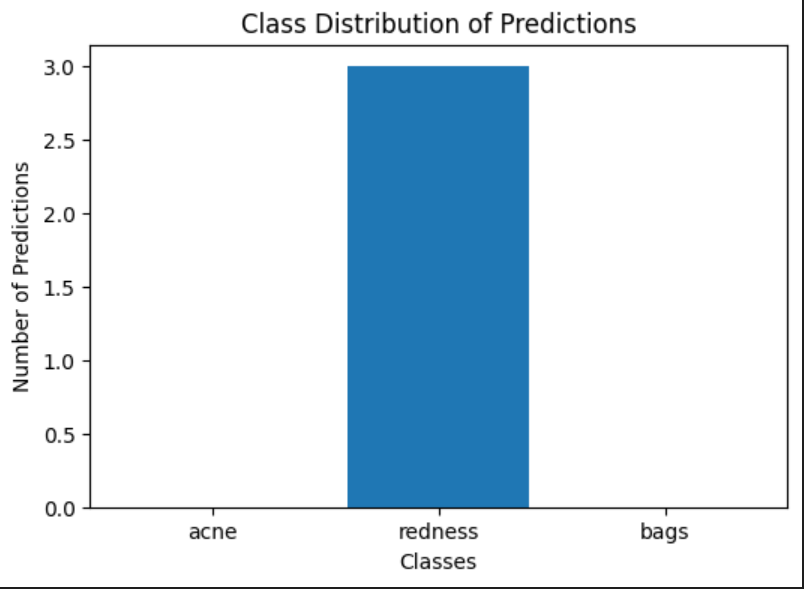
* Sample images with their true labels and predictions were displayed.



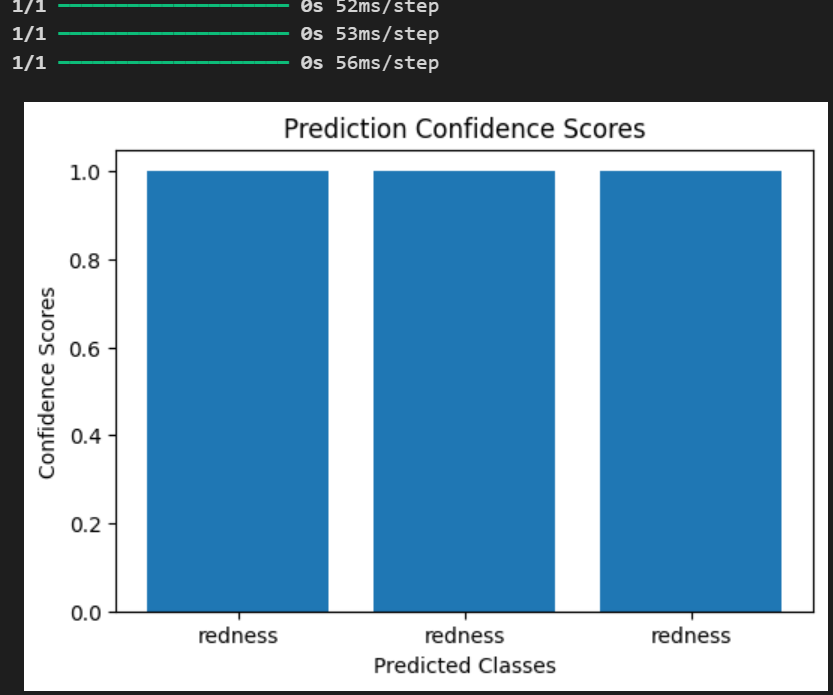
* A confusion matrix was plotted to visualize the distribution of true versus predicted labels.

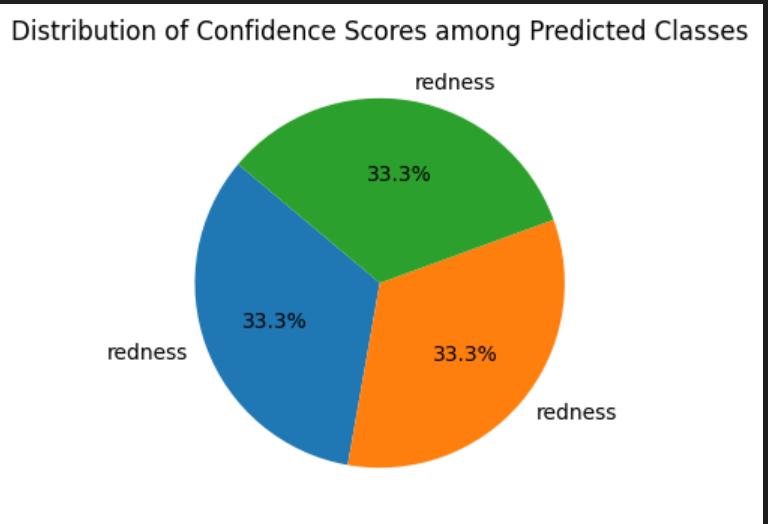


* In the confusion matrix plot, the class labels are acne, redness, and bags. The diagonal of the confusion matrix shows the number of times that the model correctly predicted the class label. For example, the top left cell of the matrix shows that the model correctly predicted acne 1 times.
* The off-diagonal cells of the confusion matrix show the number of times that the model predicted the wrong class label. For example, the cell in the second row and first column of the matrix shows that the model predicted acne 0 times for data points where the true label was redness.
* Class Distribution of Predictions



* Confidence scores of predictions were plotted as bar and pie charts.





* The pie chart above shows the distribution of confidence scores for a model that predicts the class label "redness." In other words, the pie chart slices represent different confidence levels the model has in predicting "redness" for a specific data point. It appears there are three confidence levels all at 33.3%.

**Model summary:**

****

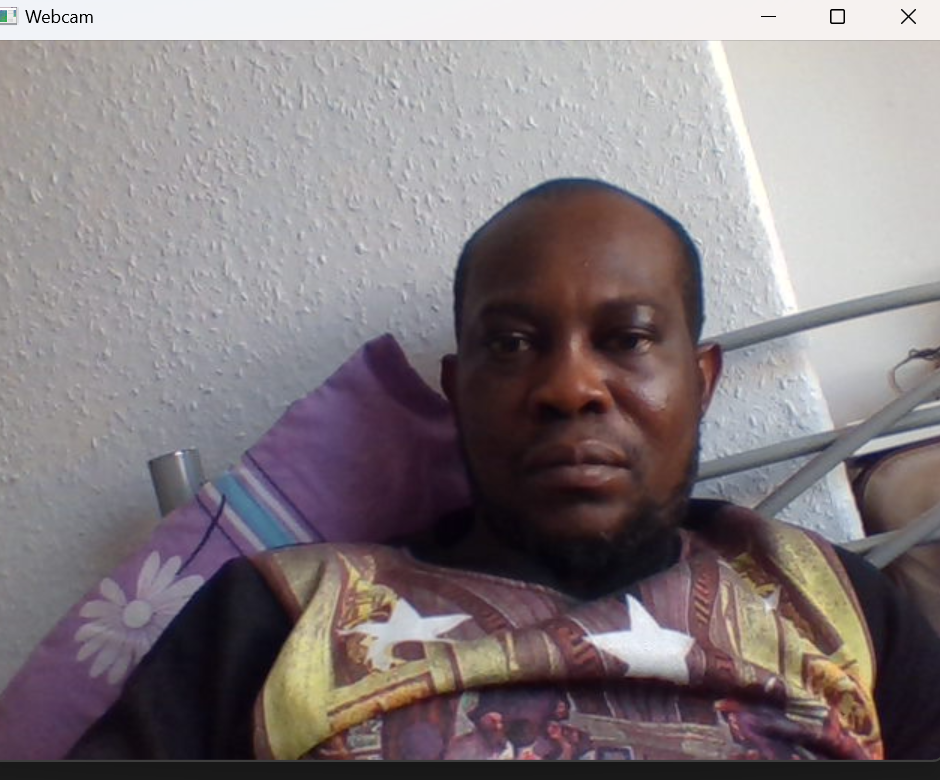
****

**4. Predictions with Live Images**

**Prediction from Webcam:**

* Images were captured using a webcam and classified using the trained model.
* # Load the trained model
* model = load\_model('best\_model.keras')
* # Define the class labels
* class\_labels = ['acne', 'redness', 'bags']  # Assuming these are the class labels in your dataset
* # Function to preprocess the image
* def preprocess\_image(img):
* img = cv2.resize(img, (img\_width, img\_height))
* img\_array = image.img\_to\_array(img)
* img\_array = np.expand\_dims(img\_array, axis=0) / 255.0  # Normalize the image
* return img\_array
* # Function to make predictions
* def predict\_image(img):
* img = preprocess\_image(img)
* predictions = model.predict(img)
* predicted\_class\_index = np.argmax(predictions[0])
* predicted\_class\_label = class\_labels[predicted\_class\_index]
* return predicted\_class\_label
* # Function to capture image from webcam
* def capture\_image():
* cap = cv2.VideoCapture(0)
* if not cap.isOpened():
* print("Error: Unable to access webcam.")
* return
* while True:
* ret, frame = cap.read()
* if not ret:
* print("Error: Failed to capture frame.")
* break
* cv2.imshow('Webcam', frame)
* if cv2.waitKey(1) & 0xFF == ord('s'):  # Press 's' to capture image
* cv2.imwrite('captured\_image.jpg', frame)
* break
* cap.release()
* cv2.destroyAllWindows()
* # Capture image from webcam
* capture\_image()
* # Read the captured image
* img = cv2.imread('captured\_image.jpg')
* # Make predictions
* prediction = predict\_image(img)
* print("Predicted class:", prediction)

**Output:**





**Challenges Faced**

**1.** **Class Imbalance**

* The dataset had an imbalance in the number of images per class, which was addressed by calculating class weights and applying them during model training.

**2.** **Data Augmentation**

* Balancing the amount of augmentation to avoid overfitting while ensuring sufficient variation was challenging.

**3.** **Model Overfitting**

* Regularization techniques such as dropout and early stopping were implemented to prevent overfitting.

**4. Data Insufficiency**

* The dataset was not large enough to train a highly accurate model. This resulted in the model often misclassifying images, for instance, predicting "redness" for images that might actually be "acne" or "bags." The limited amount of data (90 images dataset) meant that the model did not have enough examples to learn the distinguishing features of each class effectively.

**5. Real-time Prediction**

* Implementing real-time predictions using a webcam required integrating OpenCV with Keras and handling image preprocessing efficiently.

**Insights Gained**

**1. Importance of Data Augmentation**

* Data augmentation significantly improved the model's generalization ability by providing varied training samples.

**2. Model Regularization**

* Applying dropout and using L2 regularization helped mitigate overfitting and improved model performance on the validation set.

**3. Real-time Applications**

* Integrating the model with a webcam for real-time predictions demonstrated the practical applicability of the model in real-world scenarios.

**4. Need for More Data**

* To improve the model's accuracy and reliability, a larger and more diverse dataset is essential. More data would help the model learn better and differentiate between similar classes like "acne," "redness," and "bags."

**5. Visualization**

* Visualizing training progress, confusion matrices, and prediction confidence scores provided valuable insights into model performance and areas for improvement.

**Key Points on the Project Delivery:**

* **Diagnostic Accuracy**: The model aims to improve the accuracy of diagnosing skin diseases by automating the classification process using deep learning.
* **Improvement in Patient Care:** With more accurate diagnostics, patients can receive more timely and appropriate treatments.
* **Support for Dermatologists:** The model serves as a tool to aid dermatologists in making more informed decisions, potentially reducing the burden on healthcare professionals.
* **Challenges and Limitations:** The primary challenge was the limited dataset, which led to the model's inability to accurately differentiate between different skin conditions. This highlights the need for more extensive and diverse data to train more robust models.
* **Potential Impact:** With further development and a larger dataset, this model has the potential for widespread adoption in clinical settings, significantly impacting the field of dermatology and skincare.

**Conclusion**

* In conclusion, the development of this deep learning project for skin disease classification from facial images addresses the critical need for accurate and automated diagnostic tools. By leveraging state-of-the-art techniques in deep learning and computer vision, significant steps have been taken towards creating a model that can enhance diagnostic accuracy and improve patient care. Despite the challenges faced, particularly the limited dataset which impacted the model's ability to differentiate accurately between conditions like acne, redness, and bags, the project demonstrates the potential for significant advancements in dermatological diagnostics.

Overall, this project lays the groundwork for future research and development in automated skin disease classification, with the ultimate goal of improving healthcare outcomes and advancing the field of dermatology.