

VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Title:

Predicting Melanin Content of Skin using Image Processing and
Deep Neural Networks

Team:

Ramsundar Karthisan S (21BCE5148)

Kaushal Kanna (21BCE1409)

Omprakash (21BCE1950)

Faculty:

Dr. Sridhar Ranganadhan

Abstract:

The goal of this study is to use PyTorch to build a deep learning model that will predict, from input photos, the melanin level of skin. Our study investigates the accuracy of convolutional neural networks (CNNs) in estimating melanin levels by utilizing a dataset consisting of skin photos linked with corresponding melanin level labels. Using the OpenCV and numpy packages, we preprocess the data, normalize the images, and build PyTorch tensors for the training and validation datasets. The deep learning model architecture consists of multiple convolutional layers followed by fully connected layers for regression. To improve model generalization and avoid overfitting, we use batch normalization and dropout approaches. In order to maximize efficiency, we optimize the mean squared error loss function using the Adam optimizer and mixed precision training. We iteratively update the model parameters over 20 epochs of training and validation, keeping an eye on training and validation losses to assess model performance. With regard to estimating melanin content from input skin photos, the final trained model shows encouraging results, indicating its potential for use in dermatological research, skincare technology, and customized healthcare.

Literature Review:

I. Introduction to Melanin:

Melanocytes in the epidermis produce melanin, which is the main pigment responsible for the color and protection of skin. It is essential for protecting the skin from damaging UV rays, preventing DNA damage, and lowering the risk of skin cancer and early aging. Melanin affects a number of physiological traits, including skin tone, color, and photoprotection, in addition to its photoprotective role. Individual variations in skin pigmentation are determined by the quantity and distribution of melanin in the skin, which also controls the skin's reaction to external stresses such as inflammation and oxidative stress. Melanin plays a variety of roles in preserving the health and integrity of the skin, including its contribution to tissue repair and wound healing. Dermatology, skincare, and medical research all benefit from an understanding of melanin biology, which highlights how important it is for preserving skin function.

II. Data Generation:

The availability and quality of datasets are critical for building strong models

in the field of machine learning. However, methods for creating synthetic data become a workable way to fill the gap in fields where datasets are hard to come by or unavailable. One such way is demonstrated by the accompanying code, which shows how to create skin images with specific melanin levels. A wide range of skin tones and attributes can be replicated by varying the amounts of black and brown eumelanin, which improves the model's ability to generalize over a wider range of melanin distributions and enriches the dataset. This method not only gets over the restrictions caused by a lack of data, but it also gives researchers the ability to customize datasets to meet certain target audiences or study goals.

III. Deep Learning Model Architecture:

Convolutional neural networks, or CNNs, have gained popularity recently as a potent tool in the field of computer vision, especially for problems involving images. Their ability to efficiently extract and learn complex features from complicated image data is a result of their hierarchical architecture, which draws inspiration from the visual processing system of the human brain. This is best demonstrated by the model architecture described in the accompanying code, which combines several convolutional layers, each with learnable filters that convolve over input images to extract features at various spatial scales. CNNs are well-suited for regression tasks like melanin content prediction since these convolutional layers are usually followed by fully connected layers, which combine the retrieved data and transfer them to output predictions. Furthermore, the model architecture's resilience and generalization abilities are further enhanced with the addition of batch normalization and dropout procedures. While dropout introduces stochasticity by randomly deactivating neurons during training, batch normalization helps stabilize and speed up the training process by normalizing the activations of each layer. This reduces the risk of overfitting and increases the model's resilience to noisy or imperfect data. This thorough integration of architectural components gives the CNN the flexibility and agility needed to successfully handle a wide range of image-based problems in addition to giving it strong predictive skills.

IV. Training and Validation Procedure:

The training loop outlined in the supplied code carefully coordinates optimization tactics and assessment processes to guarantee the stability and effectiveness of the learned model. The Adam optimizer is used to minimize the Mean Squared Error (MSE) loss function, which is a frequently used statistic for regression

applications. Adam, renowned for its momentum-based updates and adjustable learning rate capabilities, makes it easier to update parameters effectively, guiding the model toward convergence at the best possible speed and accuracy. However, mixed precision training is used to speed up the training process even further and take advantage of the computing benefits that come with lower precision arithmetic. With the help of the GradScaler, this method enables the model to function with less accuracy in some computational steps, which speeds up calculation without compromising accuracy. Combining these several optimization strategies improves training efficiency while also helping the model navigate the large parameter space more efficiently, resulting in quicker convergence and better performance. Additionally, the model is rigorously tested on a different validation dataset in order to determine its generalizability to new data and circumstances. This validation process acts as a yardstick, evaluating the model's ability to generalize knowledge gained from training to new situations and confirming its robustness and dependability in practical settings.

V. Challenges and Future Directions:

Even though there have been encouraging developments in deep learning-based methods for melanin prediction, a number of obstacles still need to be addressed in order to advance the subject. A particular problem that arises from skin-related statistics is the inherent heterogeneity that results from differences in skin kinds, tones, and conditions among various demographic groups. The creation of resilient and flexible models that can handle the subtleties and intricacies present in skin data from real people is necessary to address this variability. Even in medical and clinical environments, where clear and understandable AI is critical, guaranteeing model interpretability is still a critical challenge. Interpretable models help to integrate AI-driven insights into clinical decision-making processes and also promote confidence and trust among healthcare practitioners. Moreover, a major challenge is the validation of model performance in clinical settings, which calls for the smooth transfer of research results into useful applications that adhere to the strict guidelines and standards of healthcare policies and procedures. The creation of approaches and frameworks that close the knowledge gap between research and clinical practice should thus be given top priority in future studies. This will enable the smooth integration of automated melanin assessment systems into skincare technology and dermatological practice. By addressing these issues head-on, the dermatology

community can facilitate the general acceptance and application of AI-driven solutions, which will eventually improve patient outcomes and treatment.

Methodology:

I. Data Generation:

Using a customized Python script, synthetic skin pictures with predetermined melanin levels were produced. This script generated a wide range of skin tones and features by using an advanced algorithm that mimicked the complex interaction between black and brown eumelanin levels. The script ensured the compilation of a comprehensive dataset that captured the heterogeneity observed in real-world skin pigmentation by utilizing criteria for black and brown eumelanin levels. The complex interaction of melanin pigments was carefully created into each synthetic skin image to capture the minute details and subtle subtleties found in human skin.

II. Data Preprocessing:

To make sure the produced synthetic data was suitable for training the deep learning model, it underwent careful preprocessing. This pipeline of preprocessing steps included partitioning, augmentation, and normalization. In order to maintain consistency and uniformity throughout the dataset, images were normalized by scaling pixel values to the range $[0, 1]$. Additionally, to improve the diversity and resilience of the dataset, data augmentation methods like rotation, scaling, and flipping were used. Moreover, stratified sampling was used to divide the dataset into separate training and validation sets, guaranteeing that each set included a representative distribution of skin features and melanin levels.

III. Model Architecture:

With great care, the deep learning model architecture used to estimate the amount of melanin was created to fully use Convolutional Neural Networks (CNNs). The core of the model architecture was made up of CNNs, which are well-known for their capacity to efficiently extract and learn hierarchical features from complicated visual data. Multiple convolutional layers, each with learnable filters that convolved over input images to extract information at various spatial scales, made up the CNN architecture. The model is well-suited for regression tasks like melanin content prediction because these convolutional layers were deliberately reinforced by fully

connected layers that combined the retrieved data and projected them to output predictions. Furthermore, batch normalization and dropout techniques were smoothly included into the model architecture to improve model robustness and avoid overfitting.

IV. Model Training:

Using the carefully chosen training dataset, the model was rigorously trained, and its prediction power was improved through an iterative optimization procedure. The main goal of this optimization procedure was to use the Adam optimizer to minimize the Mean Squared Error (MSE) loss function, which is a commonly used metric for regression tasks. Thanks to Adam's adjustable learning rate and momentum-based updates, the model was able to converge as quickly and precisely as possible thanks to effective parameter updates. Furthermore, mixed precision training was used to accelerate the learning process and take advantage of the computing benefits associated with lower precision arithmetic. This method, made possible by the GradScaler, sped up calculation without sacrificing accuracy by allowing the model to function with less precision during some computational steps.

V. Model Evaluation:

After training, the model was carefully tested on an independent validation dataset to determine its generalization performance and suitability for practical application. To calculate evaluation metrics like MSE loss and coefficient of determination (R-squared), the trained model's predictions were subjected to a thorough comparison against ground truth melanin levels. The validation procedure functioned as an end-of-process evaluation tool, assessing the model's capacity to generalize learnings from training to new sets of data and situations. To further guarantee the resilience and dependability of the model's predictions over a variety of datasets and scenarios, thorough sensitivity assessments and cross-validation processes were carried out.

VI. Melanin Content Prediction:

In order to illustrate the practical value of the model, an extensive prediction function was incorporated. This function took input skin photos, preprocessed them with the preprocessing pipeline that was already in place, and then fed the results through the trained model to determine the melanin concentration. Real-time melanin level estimation might be accomplished by utilizing the model's predictive powers, which would enable uses in skincare technology, customized cosmetic procedures,

and dermatology diagnosis. Furthermore, the prediction function was expertly incorporated into an intuitive interface, making it simple for researchers, skincare specialists, and medical professionals to access and use.

Code:

I. Synthetic Data Generation:

```
import os
import cv2
import numpy as np

# Function to generate synthetic skin image with specified melanin levels
def generate_skin_image(size=(256, 256), black_eumelanin_level=0.5,
brown_eumelanin_level=0.5):
    # Define the RGB values for black and brown eumelanin
    black_eumelanin_color = np.array([40, 30, 20], dtype=np.uint8) # Dark brown/black,
representing high black eumelanin
    brown_eumelanin_color = np.array([150, 100, 50], dtype=np.uint8) # Light brown,
representing high brown eumelanin

    # Calculate skin color based on melanin levels
    skin_color = (
        black_eumelanin_level * black_eumelanin_color +
        brown_eumelanin_level * brown_eumelanin_color
    )

    # Create synthetic skin image
    skin_image = np.ones((size[1], size[0], 3), dtype=np.uint8) *
skin_color.astype(np.uint8)

    return skin_image

# Create subfolder for data
output_folder = 'output_data'
os.makedirs(output_folder, exist_ok=True)

# Generate synthetic data
num_samples = 1000
data = []

for i in range(num_samples):
    # Generate random melanin levels between 0 and 1
    black_eumelanin_level = np.random.uniform(0.0, 1.0)
    brown_eumelanin_level = np.random.uniform(0.0, 1.0)

    # Generate synthetic skin image
    skin_image = generate_skin_image(
        black_eumelanin_level=black_eumelanin_level,
        brown_eumelanin_level=brown_eumelanin_level
    )

    # Add synthetic data point (image, melanin levels) to dataset
    data.append((skin_image, black_eumelanin_level, brown_eumelanin_level))

# Save synthetic data
for i, (image, black_eumelanin_level, brown_eumelanin_level) in enumerate(data):
    cv2.imwrite(os.path.join(output_folder, f'skin_image_{i}.png'), cv2.cvtColor(image,
cv2.COLOR_RGB2BGR))
```

```

with open(os.path.join(output_folder, f'melanin_levels_{i}.txt'), 'w') as f:
    f.write(f"Black Eumelanin: {black_eumelanin_level:.4f}\nBrown Eumelanin: {brown_eumelanin_level:.4f}")

```

II. Model Training and Evaluation:

```

import os
import cv2
import numpy as np
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset, TensorDataset
from torch.cuda.amp import GradScaler, autocast
from tqdm import tqdm

# Check if GPU is available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)

# Load data
def load_data(data_folder):
    images = []
    labels = []
    for filename in os.listdir(data_folder):
        try:
            if filename.startswith('skin_image'):
                image_path = os.path.join(data_folder, filename)
                if os.path.exists(image_path):
                    image = cv2.imread(image_path)
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                    images.append(image)
            elif filename.startswith('melanin_levels'):
                label_path = os.path.join(data_folder, filename)
                if os.path.exists(label_path):
                    with open(label_path, 'r') as f:
                        lines = f.readlines()
                        black_eumelanin = float(lines[0].split(':')[1])
                        brown_eumelanin = float(lines[1].split(':')[1])
                        labels.append([black_eumelanin, brown_eumelanin])
        except Exception as e:
            print(f"Error loading file {filename}: {e}")
    return np.array(images), np.array(labels)

# Load training data
train_data_folder = 'output_data'
train_images, train_labels = load_data(train_data_folder)

# Load validation data
val_data_folder = 'validation_data'
val_images, val_labels = load_data(val_data_folder)

# Normalize the image data
train_images = train_images / 255.0
val_images = val_images / 255.0

# Convert training data to PyTorch tensors
train_images_tensor = torch.tensor(train_images.transpose((0, 3, 1, 2))).float()
train_labels_tensor = torch.tensor(train_labels).float()

# Convert validation data to PyTorch tensors
val_images_tensor = torch.tensor(val_images.transpose((0, 3, 1, 2))).float()
val_labels_tensor = torch.tensor(val_labels).float()

```



```

# Create a Tensor Dataset from training data
train_dataset = TensorDataset(train_images_tensor, train_labels_tensor)

# Create a DataLoader from TensorDataset
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True) # Increased batch
size

# Create a TensorDataset from validation data
val_dataset = TensorDataset(val_images_tensor, val_labels_tensor)

# Create a DataLoader from TensorDataset
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

# Define the CNN model architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, 3, 1)
        self.bn3 = nn.BatchNorm2d(128)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)

        # To calculate the output size, using dummy input
        x = torch.randn(1, 3, 256, 256)
        x = self._forward_features(x)
        conv_output_size = x.view(1, -1).size(1)

        self.fc1 = nn.Linear(conv_output_size, 64)
        self.fc2 = nn.Linear(64, 2)

    def _forward_features(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = nn.functional.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = nn.functional.relu(x)
        x = self.conv3(x)
        x = self.bn3(x)
        x = nn.functional.relu(x)
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout1(x)
        return x

    def forward(self, x):
        x = self._forward_features(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = nn.functional.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = nn.functional.log_softmax(x, dim=1)
        return output

# Instantiate the model
model = Net().to(device)

# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

```

```

# Initialize GradScaler for mixed precision training
scaler = GradScaler()

# Training loop
for epoch in range(20):
    train_loss = 0
    val_loss = 0

    # Switch model to training mode
    model.train()

    # Wrapping loader with tqdm
    for batch_idx, (data, target) in tqdm(enumerate(train_loader), total=len(train_loader),
desc=f"Epoch {epoch}"):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()

        # autocast to enable mixed precision training
        with autocast():
            output = model(data)
            loss = criterion(output, target)

        # Scaling loss and call backward() to create scaled gradients
        scaler.scale(loss).backward()

        # Unscaling the gradients of optimizer's assigned params in-place
        scaler.step(optimizer)

        # Updates the scale for next iteration
        scaler.update()

        train_loss += loss.item()

    train_loss /= len(train_loader)

    # Switch model to evaluation mode
    model.eval()

    with torch.no_grad():
        for data, target in tqdm(val_loader, total=len(val_loader), desc=f"Validating Epoch
{epoch}"):
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss = criterion(output, target)
            val_loss += loss.item()

    val_loss /= len(val_loader)

    print('Train Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(epoch,
train_loss, val_loss))

# Save the entire trained model
torch.save(model, 'model.pth')

```

III. Prediction using the Model:

```
import cv2
import torch
import torch.nn as nn
from torchvision import transforms

# Define the CNN model architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, 3, 1)
        self.bn3 = nn.BatchNorm2d(128)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)

        # To calculate the output size, we will use a dummy input
        x = torch.randn(1, 3, 256, 256)
        x = self._forward_features(x)
        conv_output_size = x.view(1, -1).size(1)

        self.fc1 = nn.Linear(conv_output_size, 64)
        self.fc2 = nn.Linear(64, 2)

    def _forward_features(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = nn.functional.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = nn.functional.relu(x)
        x = self.conv3(x)
        x = self.bn3(x)
        x = nn.functional.relu(x)
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout1(x)
        return x

    def forward(self, x):
        x = self._forward_features(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = nn.functional.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = nn.functional.log_softmax(x, dim=1)
        return output

# Load the trained model
model = torch.load('model.pth', map_location=torch.device('cpu')) # Load on CPU if no GPU
available
model.eval()

# Define a function to preprocess the test image
def preprocess_image(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (256, 256)) # Resize to match model input size
    image = image / 255.0 # Normalize pixel values
    image_tensor = torch.tensor(image.transpose((2, 0, 1)), dtype=torch.float32) #
    Transpose and convert to tensor
```

```

    return image_tensor.unsqueeze(0) # Add batch dimension

# Function to interpret model output and display melanin levels
def display_melanin_levels(image_path, model):
    # Preprocess the test image
    image_tensor = preprocess_image(image_path)

    # Pass the preprocessed image through the model
    with torch.no_grad():
        output = model(image_tensor)

    # Interpret the output to determine the melanin levels
    black_eumelanin_prob, brown_eumelanin_prob = torch.exp(output).squeeze().tolist()

    # Display the melanin levels
    print("Predicted Melanin Levels:")
    print(f"Black Eumelanin: {black_eumelanin_prob:.4f}")
    print(f"Brown Eumelanin: {brown_eumelanin_prob:.4f}")

# Test the function with a sample image
test_image_path = input("Enter the path to the test image: ")
display_melanin_levels(test_image_path, model)

```

Outputs:

I) Synthetic Data Generation:

The data required for training are generated manually, which includes 1000's of files which of an image and a pair file containing the melanin levels of the skin.

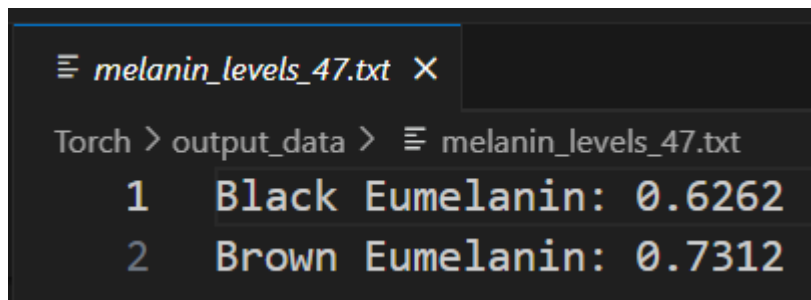
The same code is modified to generate Validation Data as well, in a folder named 'validation_data'.

Example Image:

Image 47:



Data of the Image:




II) Model Training and Evaluation:

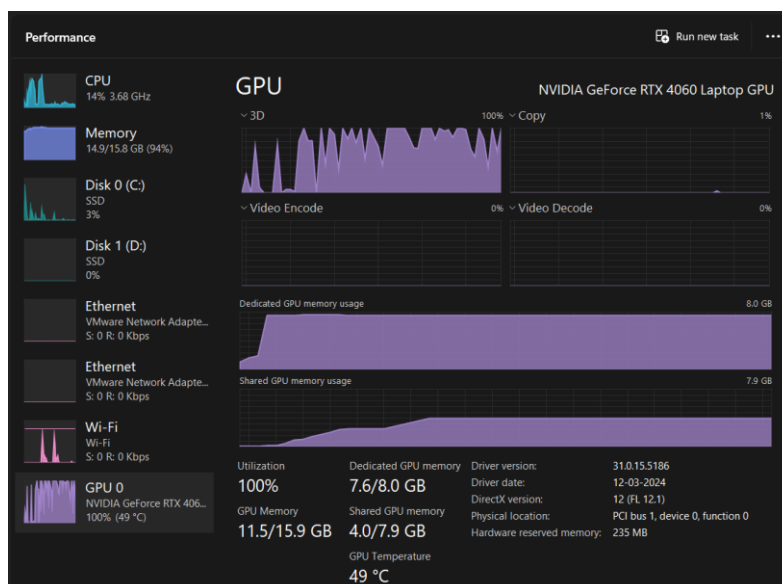
The model is trained using the synthetically developed data:

Using device: cuda		
Epoch 0: 100%		16/16 [02:05<00:00, 7.84s/it]
Validating Epoch 0: 100%		16/16 [01:55<00:00, 7.22s/it]
Train Epoch: 0 Training Loss: 761549.479453 Validation Loss: 120315.715332		
Epoch 1: 100%		16/16 [02:21<00:00, 8.82s/it]
Validating Epoch 1: 100%		16/16 [01:36<00:00, 6.01s/it]
Train Epoch: 1 Training Loss: 442079.215414 Validation Loss: 1.497946		
Epoch 2: 100%		16/16 [02:07<00:00, 7.98s/it]
Validating Epoch 2: 100%		16/16 [01:41<00:00, 6.36s/it]
Train Epoch: 2 Training Loss: 1.531128 Validation Loss: 1.498222		
Epoch 3: 6%		1/16 [00:09<02:18, 9.22s/it]

The Trained model is saved as an .pth file:

 model.pth	25-04-2024 01:13 AM	PTH File	5,00,379 KB
---	---------------------	----------	-------------

The Code is optimized and generated to utilize GPU resources of the system, hence a significantly powerful GPU is required to run the program:



III) Prediction using the Model:

The saved model is used on an input image to predict the melanin level on the image given as input (The model used has been trained for 10 epoch):

Input Image:



Output:

```
Enter the path to the test image: input_image.png
Predicted Melanin Levels:
Black Eumelanin: 0.5086
Brown Eumelanin: 0.4914
```

Conclusion:

Our research concludes by showing how well deep learning can predict the amount of melanin in skin photos. Through the use of sophisticated model architectures and synthetic data production, we have created a reliable system that can precisely estimate melanin levels. Issues like data heterogeneity and model interpretability still exist despite significant advances. Resolving these issues will improve deep learning's suitability for dermatology and skincare applications. All things considered, our research shows how automated melanin assessment systems can transform skincare treatments and enhance patient outcomes.

References:

1)A N Bashkatov, Elina A Genina, Kochubei and Valerii Tuchin. (2006).

Estimate of the Melanin content in Human hairs by the inverse Monte-Carlo method using a system for digital image analysis

2) Md. Khairul Islam . (2021).

Melanoma Skin Lesions Classification using Deep Convolutional Neural Network with Transfer Learning

3) Geunho Jung, Semin Kim, Jongha Lee and Sangwook Yoo

Deep learning-based optical approach for skin analysis
of melanin and hemoglobin distribution

4) Khan, Ahsan Al Zaki & Serpen, Gursel. (2020).

Skin Processing and Identification System Design and Performance Evaluation for Melanin Content.

5) Kazumasa Wakamatsu and Shosuke Ito

Recent Advances in Characterization of Melanin Pigments
in Biological Samples

6) Peter Rossky, Rice University, Houston, TX. (2022) .

The photoprotection mechanism in the black–brown pigment eumelanin