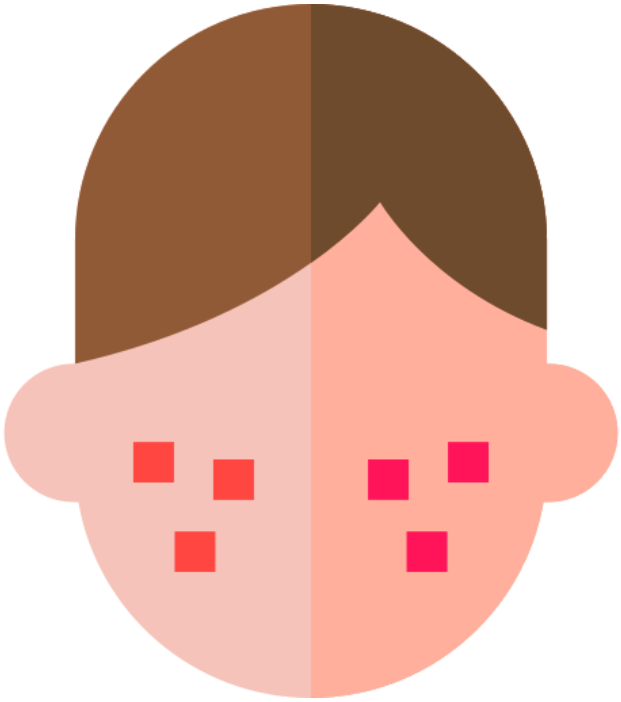


Acne Detection and its Removal

28.04.2024

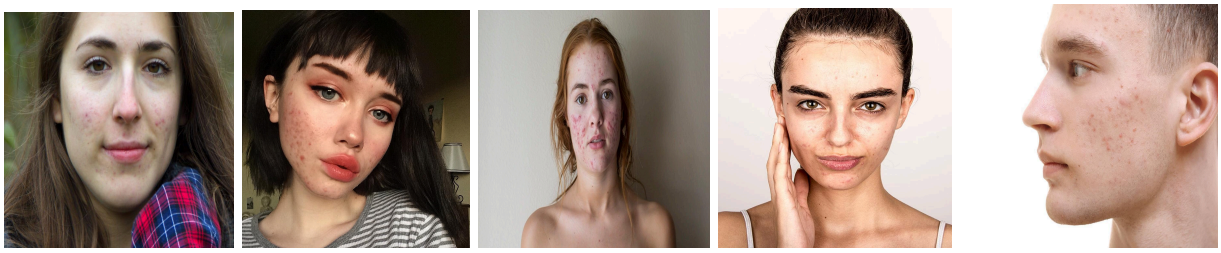
Prof. Rajendra Nagar and Prof. Pratik Mazumder
Nakkina Vinay (B21AI023)
Geda Durga Vara Praveen (B21CS031)
Marri Bharadwaj (B21CS045)
Rankireddy Sai Mani Akarsh (B21CS063)

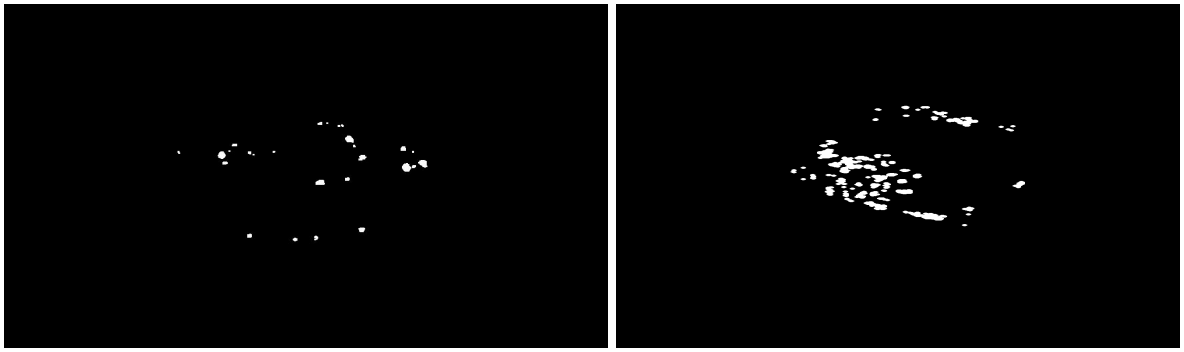
Problem Statement

One of the most common skin conditions, acne, affects most people at some point in their lives. But in the present era, society values having healthy, attractive skin, thus many individuals utilise various applications/filters to digitally erase pimples from their photos before sharing them on social media. Thus, we have used computer vision techniques to create a pipeline that helps detect and remove acne from their photographs. To achieve this, several techniques have been used, such as thresholding, masking, blurring, etc.

Dataset

We've used the Custom Dataset for this project. The dataset contains facial images and their corresponding masks are available. It consists of 30 images and corresponding acne masks.



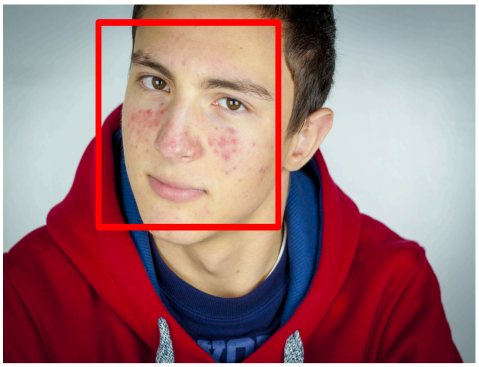


Methodology

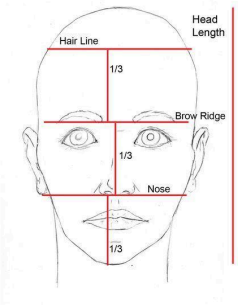
Face Detection

To detect the face from the image, we've used two methods:

- ***DLIB* library:** It works by initialising a face detector dlib.get_frontal_face_detector(), at first. It's a pre-trained model that is capable of predicting the locations of 68 facial landmarks in an image. Then, we detected faces in the concerned grayscale image by gathering the rectangles where each rectangle represents the bounding box of a detected face. For each detected face, we used this shape predictor dlib.shape_predictor() technique to predict the locations of 68 facial landmarks defining the pose of the object.

		
Original image	**Face detected**	**Key Points detected**

- ***Face Alignment Network (FAN)*:** We initialised the face alignment model face_alignment.FaceAlignment() and predicted the 2D facial landmarks for each detected face in the image. Additionally, we found 3 forehead points and a total 71 points were computed. After that, we computed the convex hull using multi_point.convex_hull() of the facial landmarks to help in understanding the overall shape and structure of the face. The convex hull represents the outer boundary of the facial landmarks, which can be useful for tasks like face recognition.

			
Original image	**Facial points Detected**	**Eg. for additional points generation**	**Key Points along with forehead points**

Acne detection - Method 1: Thresholding Based on Skin Colour

Face region selection:

We've taken the convex hull points generated for the image and filled the convex hull of facial landmarks with white colour and masks. The resulting masked image is displayed, showing only the facial region with noise removed and boundaries refined.

Markuping redundant facial regions:

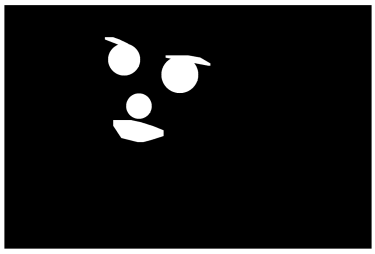
We segmented the facial points using landmark indices and extracted them based on the image and its alignment. For example, if the face was aligned to the right, we'd take the right eyebrow, right eye, in addition to the nose and mouth, and increase the threshold and make it bright, while the remaining would be dark.

Estimating average skin colour in *A* channel of LAB colour scheme:

We then converted the *RGBimage* to *Lab colour* space *(L (lightness)*, *A (green to magenta)*, and *B (blue to yellow))* cv2.cvtColor(acne_mask, cv2.COLOR_RGB2LAB) for analysing the channel *A*. Acne lesions often appear as reddish or pinkish spots on the skin due to inflammation and increased blood flow. These colours fall within the *magenta* range in the *A* channel, which represents a spectrum from green to magenta in the *LAB colour* space. Then, we calculated a threshold value based on the mean intensity of the *A* channel within the highlighted acne areas. This threshold value was used to segment the acne areas from the background.

Morphological Operations:

We defined a *3x3 kernel* for morphological operations. cv2.morphologyEx(A_th, cv2.MORPH_CLOSE, kernel, iterations=1) performed a morphological closing operation on the thresholded image to fill in any small holes or gaps in the acne regions. We dilated the thresholded image to further enhance and smooth the acne regions.

			
Face Region selection	**Markuping Redundant Facial regions**	**Estimating Average skin colour in *A* channel of *LAB colour* scheme**	**Mask creation by Thresholding**

Acne detection - Method 2: Thresholding Based on Difference (A-GB)

Finding the possible acne regions:

We found the image *img_cnt_1* that represented the regions of interest (facial features) in the image. We then converted the *imageRGB* into *LAB* space and extracted the *A* channel from that which suited well in detecting acne. The feature (*A* in this case, likely representing the *A* channel of the *LAB colour* space) underwent *Gaussian blurring (GB)*. *Gaussian blurring* is a common technique used to smooth images by reducing high-frequency noise and detail. It replaces each pixel's value with a weighted average of its neighbouring pixels' values, using a *Gaussian kernel*. After that, we had calculted the difference between the original feature and a *Gaussian blurred* version *(A - GB)*. This likely helped in highlighting certain patterns or anomalies in these features.

Thresholding on Intensity Value:

We then applied thresholding *(diff > 42)* and morphological operations (dilation) to isolate regions that potentially contain acne. Thresholding is a common technique to separate

regions of interest based on intensity differences. Morphological operations help in refining the detected regions and filling in small gaps.

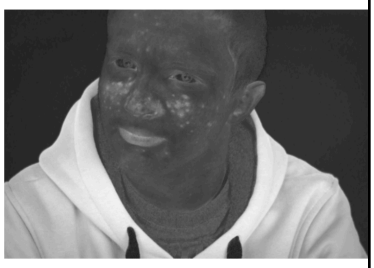
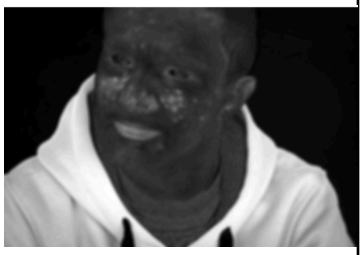
	

Image in *A* channel of *LAB colour* scheme (1)	**Applying *Low Pass 2D Gaussian* filter (2)**	**Difference between (1) and (2)**	**Mask creation by thresholding**

Estimate All (function to process image + metric calculations)

We calculated the recall score using 2 methods:

- *face_alignment.Face_alignment* used to find face alignment.
- *Multipoint.convex_hull* used to find shape of face

In this step, we calculated the recall score for the dataset we were using using the two methods. *Method 1* converted *RGBImage* to *Lab colour* image, and found the acne by increasing the threshold based on skin colour. *Method 2* calculated the difference between *RGBImage* and *Guassian filter* image and then found a mask using this.

In the dataset, we had existing masks related to images and we found masks for these images using the above methods, and thus found recall score using existing mask and calculated mask.

By comparing two method results and finding the mean, we got maximum for *Method 2*:

```
scores_1 = np.array(scores_1)
scores_2 = np.array(scores_2)

print(np.mean(scores_1))
print(np.mean(scores_2))
```

```
0.4420409534264906
0.49792804759217424
```

Inpainting (Acne Removal)

Implemented two methods for inpainting:

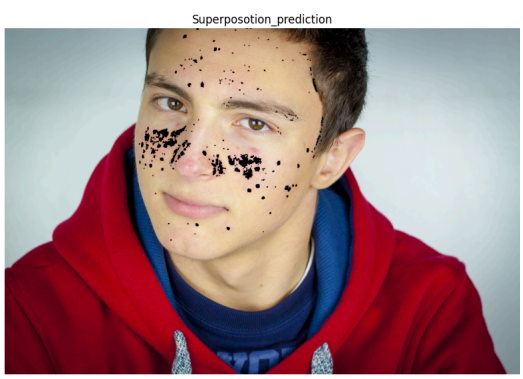
- *cv2.inpaint* using *cv2.INPAINT_TELEA* described in the paper
- Application of Deep Image Prior to image restoration

5.1 – Inpainting (Acne Removal) - Method 1: *OpenCV* Integrated Painting

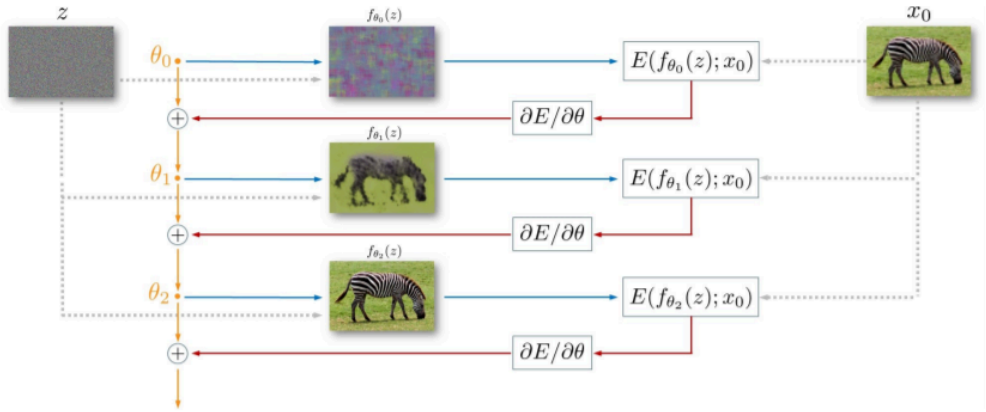
This method was aimed at removing the acne from facial images using *OpenCV*.

Inbuilt function *cv2.inpaint()* took the input image which was the original image, *acne_mask* (generated in the pipeline before) and the inpainting radius which determined how large of a region around each damaged pixel should be considered for the inpainting process. *cv2.INPAINT_TELEA* is a flag specifying the inpainting algorithm to use.

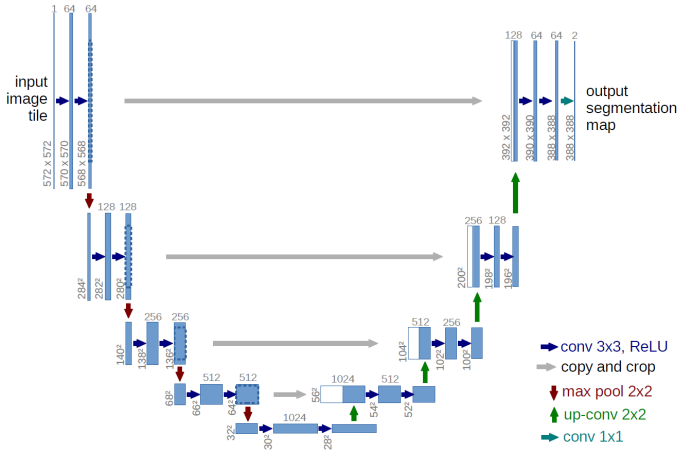
After inpainting, by using the *cv2.imwrite()* to save the *inpainted* image; we plotted the image:

		
Original image	**Image with Mask Generated**	**Impainted Image Using OpenCV**

5.2 – Inpainting (Acne Removal) - Method 2: Deep Image Prior



Deep Image Prior procedure f - Neural network (Unet)



Unet Architecture

First, we implemented the model architecture using the *Deep Convolutional Neural Networks*. Instantiating an instance of the *Unet* model with a dropout probability of 0.05. Dropout is a regularisation technique used to prevent overfitting by randomly dropping a certain percentage of neurons during training.

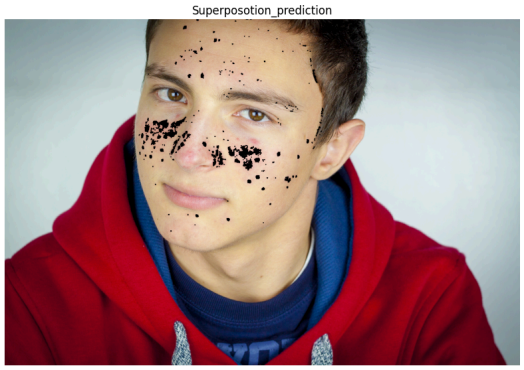
Using the *Mean Squared Error Loss* function, which is commonly used for regression tasks, we calculated the mean squared difference between the model's prediction and the ground truth labels.

We used *Adam Optimisation* Algorithm for updating the model parameters during training. *Adam* is a popular optimization algorithm known for its efficiency and effectiveness in training deep neural networks. The Learning rate of 0.01 was used.

Then using the *train* function for training the model. The function iterated through the specified number of epochs (5000 here). Within each epoch, the model is used to generate an output image based on the random input *z*. The loss was calculated based on the generated output and the target image multiplied by the mask. The *train* function then returned the final prediction and a list of predicted images.

After training the model, we visualised as follows:

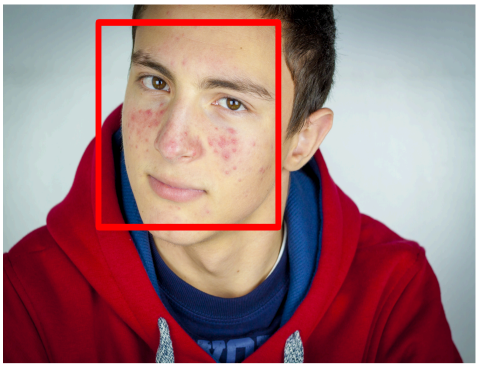
The Final Prediction

		
Original image	**Image with Mask Generated**	**Impainted Image Using Unet**

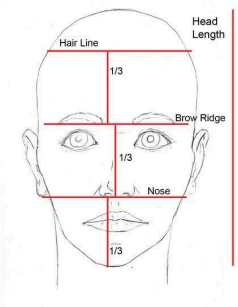
Results

Face Alignment

1. *dlib*

		
Original image	**Face detected**	**Key Points detected**

2. *Face Alignment Network (FAN)*

			
Original image	**Facial points Detected**	**Eg. for additional points generation**	**Key Points along with forehead points**

Acne Detection

1. *Threshold based on Skin Colour*

			
Face Region selection	Markuping Redundant Facial regions	Estimating Average skin colour in *A* channel of *LAB colour* scheme	Mask creation by Thresholding

2. *Threshold based on difference in Light colour Image and Gaussian Blur Image*

			
Image in *A* channel of *LAB colour* scheme (1)	Applying *Low Pass 2D Gaussian* filter (2)	Difference between (1) and (2)	Mask creation by thresholding

Estimating Mean for recall scores for 2 models for different Images and Masks

1	0.681	0.573
2	0.95	0.619
3	0.458	0.531
4	0.634	0.774
5	0.131	0.148
6	0.353	0.597
7	0.075	0.07
8	0.44	0.868
9	0.27	0.667
10	0.693	0.465
11	0.771	0.683
12	0.362	0.304
13	0.685	0.567
14	0.611	0.715
15	0.554	0.545
16	0.442	0.448
17	0.333	0.547
18	0.493	0.6
19	0.534	0.608
20	0.239	0.305
21	0.358	0.3
22	0.479	0.617
23	0.475	0.493
24	0.117	0.06
25	0.126	0.12
26	0.202	0.519
27	0.332	0.494
28	0.668	0.798
29	0.373	0.38
30	0.423	0.52

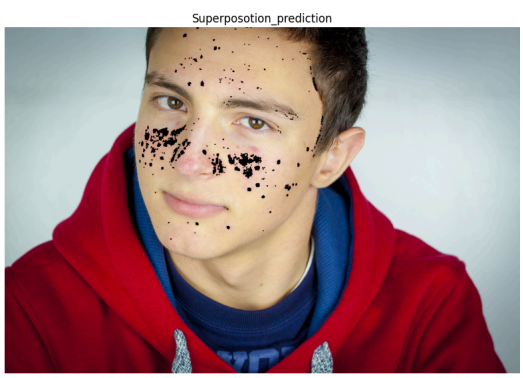
```
scores_1 = np.array(scores_1)
scores_2 = np.array(scores_2)

print(np.mean(scores_1))
print(np.mean(scores_2))
```

```
0.4420409534264906
0.497928047592174 24
```

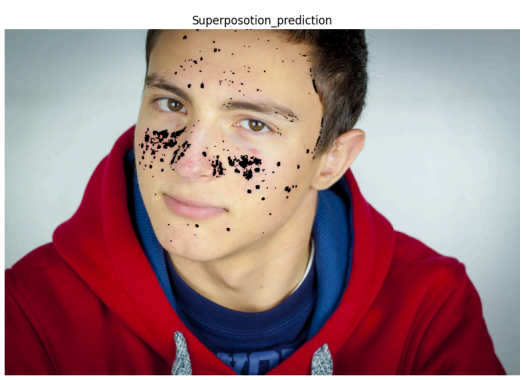
Based on the results, *Method 2* (Threshold based on difference in *A* - *GB*) had a high mean.

Inpainting (Acne Removal)

OpenCV integrated painting

		
Original image	**Image with Mask Generated**	**Impainted Image Using OpenCV**

Deep Image Prior

		
Original image	**Image with Mask Generated**	**Impainted Image Using Unet**

Observations and Conclusions

We calculated the recall score using the dataset using *Method 1* and *Method 2*. And we found that *Method 2* is best for finding masks. So we used inpainting for results which we got from *Method 2*.

```
print(np.mean(scores_1))
print(np.mean(scores_2))
```

```
0.4420409534264906
0.497928047592174 24
```

References

https://www.researchgate.net/publication/238183352_An_Image_Inpainting_Technique_Based_on_the_Fast_Marching_Method

https://github.com/1adrianb/face-alignment

http://dlib.net/python/index.html#dlib.shape_predictor

http://dlib.net/python/index.html#dlib.get_frontal_face_detector

https://arxiv.org/pdf/1703.07332

https://docs.opencv.org/4.x/

https://docs.opencv.org/3.4/df/d3d/tutorial_py_inpainting.html