Final Report - Automatic Photodamage assessment from Facial Images

Fall 2020 Capstone Project - Unilever
Dec 18, 2020

Team Members
Mingrui Liu (ml4404)
Shuyu Huang (sh3967)
Woo Jin Kim (wk2340)
Yuheng Shi (ys3172)
Yunjun Xia (yx2569)

Mentor
Robert Velthuizen

Advisor
Adam S. Kelleher

Final Report - Automatic Photodamage assessment from Facial Images **1**
Abstract **3**
1 Introduction **3**
2 Exploratory Data Analysis and Data Preprocessing **4**
Dataset 4
Grade Analysis 5
Grade Grouping 5
Input Pipeline with Tensor 5
Train, Validation, and Test Set Splits 6
Data Augmentation 6
Normalization 6
3 Modeling **6**
Baseline Model 6
Method 1: Image Subtraction and Binary Threshold 8
Model Construction and Results 8
Potential Problems 10
Method 2: Siamese Network 10
4 Conclusion **15**
5 Future Work **15**
Collect More Data With Extreme Grades 15
Improve Baseline Model with Other Transfer Learning Model 15
Crop Facial Image to Extract Regions of Interest More Precisely 15
References **16**

Abstract

The capstone project with Unilever aims to simulate an auto-grader for grading mottled hyperpigmentation on the facial images. In this report, we include exploratory data analysis on the image dataset. Also, we discussed data input pipeline with tensor, and data preprocessing procedures, which include manual train, validation, and test set splits, data augmentation, and normalization. Then we discussed three deep learning models: transfer learning with ResNet50, transfer learning with images after binary thresholding, and Siamese neural networks. Our best practice approaches test RMSE as low as 0.39 and test accuracy as high as 81.25%. In the end, we discussed potential improvements and future work.

1 Introduction

This capstone project is a collaboration between Columbia University Data Science Institute and Unilever. The objective of this project is to build a model by leveraging deep learning and image analysis to obtain automatic grading on mottled hyperpigmentation conditions from facial images taken in clinical studies. In such studies, subjects' faces are treated, where different treatments may have different average effects as seen by an expert grader.

The need for automating the judgments arises because of

(1) Limited availability of expert graders
(2) Difference between graders
(3) Some subjectivity and variability of the experts

Previous attempts by Unilever to build multivariate models on image analysis parameters were not successful. Similarly, deep learning with VGG16 transfer learning did not yield the desired outcome.

Our goal is that other data science methods, such as a knowledge-based approach or combinations of techniques may have the potential to result in the desired outcome. With a given dataset of 2403 valid image-grade pairs, we used three different models in our project to compare which model yields the desired outcome.

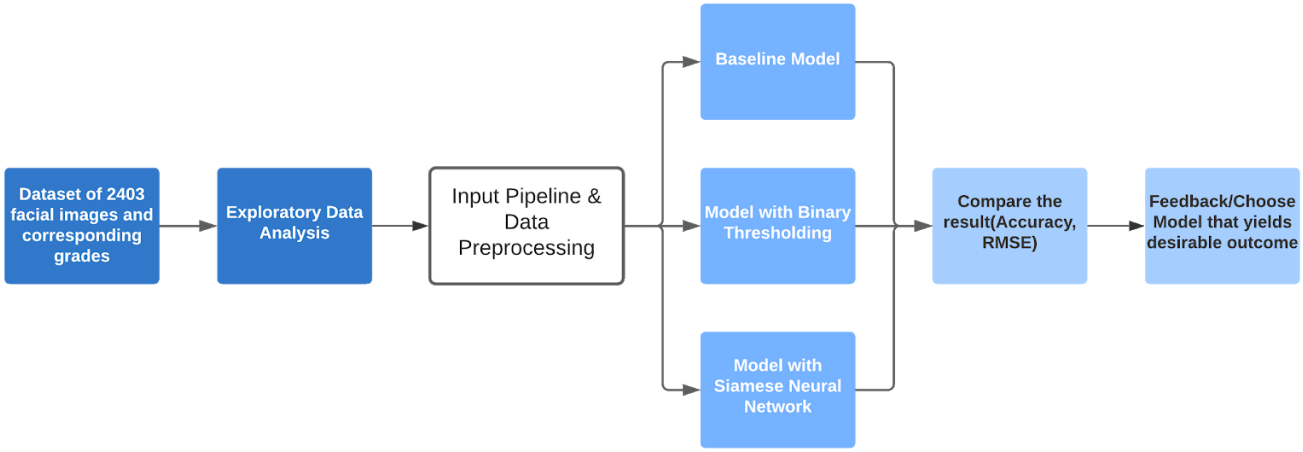

Figure 1.1 Flowchart of the Project

2 Exploratory Data Analysis and Data Preprocessing

Dataset

We have 3 different types of data, images (.jpg), grades (.xlsx), and grading criteria (.docx).

1. Image data (.jpg)
   We have 2403 images. As the sample image below (Figure 2.1), each JPEG file contains one side of a person's face. For privacy restriction, we obtained images, which only retained skin and masked out other characteristics such as hair, eyebrows, eyelashes, nostrils, and mouth. We can clearly observe that there is some hyperpigmentation area on the image below. For example, there is a darker area above the eyebrow and several darker spots on the cheek.

Figure 2.1 sample image (.jpg)

2. Grades for each image

	SUBJID	side	VISITNUM	VISIT	Mottled hyperpigmentation
2	1	Left	1	Baseline	2.5
3	1	Left	2	Week 5	2.5
4	1	Left	3	Week 8	2
5	1	Left	4	Week 12	1.5
6	1	Left	5	Week 16	1.5

Figure 2.2 Grades

3. Grading Criteria
   The grading criteria explains how the skin condition is graded. Our goal is to simulate an auto-grader that grading skin condition per grading criteria. According to the grading criteria (Figure 2.3), the grade ranges from 0 to 9 with an interval of 0.5. More severe hyperpigmentation will result in a higher grade. We can group hyperpigmentation conditions to none, mild, moderate, and severe. This is one of the basis we do grouping in the following grade analysis and grouping part.

Mottled Hyperpigmentation (Entire Side of Face)				
Grade				**Description**
0			None	No hyperpigmentation
1	2	3	Mild	None/very few small or faint brown patches not obvious from a distance
4	5	6	Moderate	Obvious presence of brown spots, which may be faint but covering the entire face, or very pronounced but more localized
7	8	9	Severe	Severe –moderately/severely pigmented brown patches often with whitish spots (mottled). Very noticeable from a distance

Figure 2.3 Grading criteria

Grade Analysis

From the distribution of grades in our dataset (Figure 2.4), we observed that the distribution of photodamage grade in our dataset is imbalanced. As shown in Figure 2.4 (left), we found that the number of images with grades less

than 3.0 and the number of images with grades larger than 5.0 is much smaller than the number of images with grades between 3.0 to 5.0. Therefore, the dataset is imbalanced, which would result in less accuracy when predicting facial photodamage with extreme patterns.

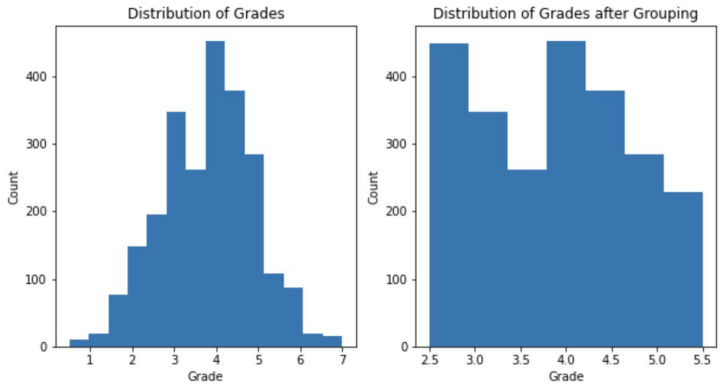

Figure 2.4 The distribution of grades before (left) and after (right) grouping

Grade Grouping

The method we used to solve the problem of the imbalanced dataset is to wrap up the images with extremely low and high grades into two categories. That is, we created a category named `<3.0` containing all images with grades less than 3.0, and another category named `>5.0` containing all images with grades greater than 5.0. Then we would have 7 classes instead of 14: `<3.0`, `3.0`, `3.5`, `4.0`, `4.5`, `5.0`, `>5.0`. The following tables (Figure 2.5) show the number of images corresponding to different classes before and after subgrouping. After aggregating 14 classes to 7 classes, though the number in each class is still not strictly equal (Figure 2.4, right), we can assume that they are evenly distributed.

Grade	Image count
0.5	10
1.0	19
1.5	76
2.0	148
2.5	196
3.0	348
3.5	261
4.0	452
4.5	379
5.0	285
5.5	108
6.0	87
6.5	18
7.0	16

→

Grade	Image count
< 3.0	449
3.0	348
3.5	261
4.0	452
4.5	379
5.0	285
> 5.0	229

Figure 2.5 Tables of number of images before and after grouping

Input Pipeline with Tensor

We chose to treat images as tensors and applied the technique of caching and prefetching to maximize efficiency. Using caching, the transformation before the cache operation will be executed once at the first epoch. Then we could reuse the data cached in the following epochs. With prefetching, the input pipeline will pre-read the data for the next step while the current step is running. With tensors and these two methods, the time spent on each training epoch is reduced from 80 seconds to 15 seconds when training the baseline model, which is around 70% faster than the generator method.

Train, Validation, and Test Set Splits

For the cross-validation steps in the future model evaluating part, we prepared the train, validation, and test set splits manually. The reason is that because of the nature of caching and prefetching, data would be erased from local memory if it exceeds the capacity, and not all images are loaded at the same time.

Data Augmentation

Since a great amount of data is required in training parameters of the transfer learning model, we chose to randomly flip images horizontally to amplify the dataset and increase the amount of relevant data. The strategy of horizontal random flip can also help to avoid overfitting.

Normalization

After data augmentation, we also chose to normalize the augmented dataset prior to training all of the following models. By doing so, we converted the image pixels to range from -1 to 1, as the original pixels range from 0 to 255. The normalization enables neural networks to work with data easier by improving computation efficiency.

3 Modeling

Baseline Model

After the preprocessing steps, we chose to implement Transfer Learning. As mentioned in the previous section, our region of interest is the mottled hyperpigmentation, which is the detailed information on faces. Among the set of Transfer Learning models, *ResNet50* performs better on detecting image details instead of image shapes or edges. Thus, we used *ResNet50* as our baseline model.

ResNet50 is pre-trained on *ImageNet*. There are 175 layers in *ResNet50* in *Keras* Applications. These 175 layers include 50 convolutional layers, and other layers like batch normalization layers, activation layers, pooling layers, etc. For better model performance, we set the first 160 layers in *ResNet50* to be non-trainable, and the rest 15 layers to be trainable. By de-freezing some layers in the transfer learning model, we can focus more on the detailed mottled hyperpigmentation areas since the base model will be partially trained on our image dataset. Then, we added one drop-out layer with a rate of 0.5 and one global average pooling layer. These two layers can help to reduce overfitting. The drop-out layer will randomly drop out neurons in the neural network, and the global average pooling works as a dimensionality reduction method which will help to reduce the total number of parameters in the model. We also applied a dense layer with *Softmax* activation function as our classifier layer. Then, we used *Adam* optimizer with *Exponential Cyclical learning rate*. This method enables the learning rate to cyclically vary between reasonable boundary values. We set the learning rate to vary from 1e-8 to 1e-4 with step size 240. Also, using the *Exponential Cyclical learning rate* can help to practically eliminate the need to tune the best learning rate. Moreover, we used *Sparse Categorical Cross-Entropy* as the loss function and the default *Accuracy* as metrics. Figure 3.1 shows the construction summary of our baseline model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 16, 16, 2048)      23587712
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 2048)      0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 14343
=================================================================
Total params: 23,602,055
Trainable params: 5,534,727
Non-trainable params: 18,067,328
```

Figure 3.1 Baseline Model Summary

We trained 50 epochs with batch size 32 and shuffle buffer size 1024. Figure 3.2 and Figure 3.3 show that we achieved 99.94% in training accuracy and around 52% validation accuracy. From the test data, we achieved 50.42% accuracy.

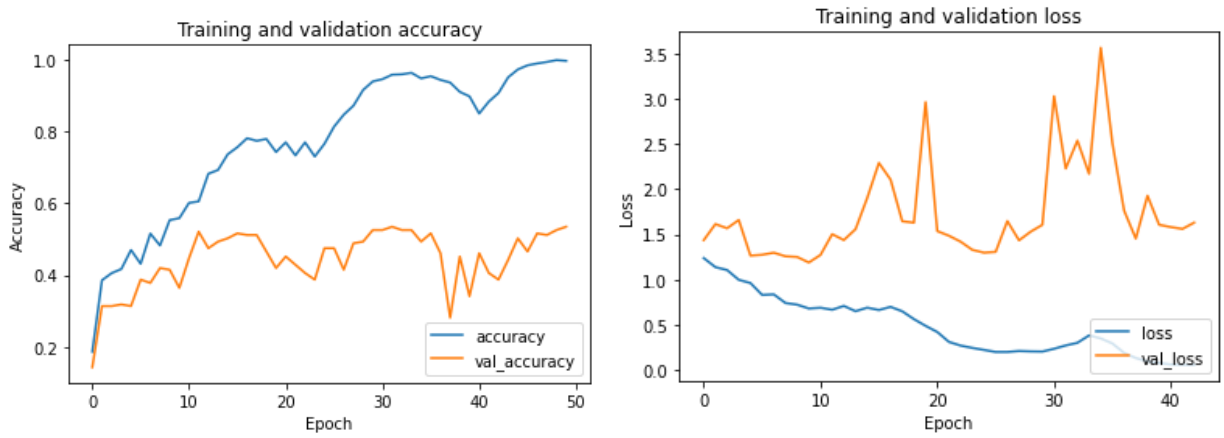

Figure 3.2 Training and validation accuracy and loss for the baseline model

```
[ ] loss, accuracy = model.evaluate(test_ds)
    print("Accuracy: {:2.2%}".format(accuracy))

    19/19 [==============================] - 5s 256ms/step - loss: 2.4354 - accuracy: 0.5042
    Accuracy: 50.42%
```

Figure 3.3 Test accuracy for the baseline model

According to the confusion matrix, shown in Figure 3.4, we found that most of the misclassifications are near the diagonal. That is, whenever misclassification occurs, our predicted label is not that far away from the true label. For example, images in group 3 (represents actual grade 4) are mostly misclassified as group 4 (represents actual grade 4.5).

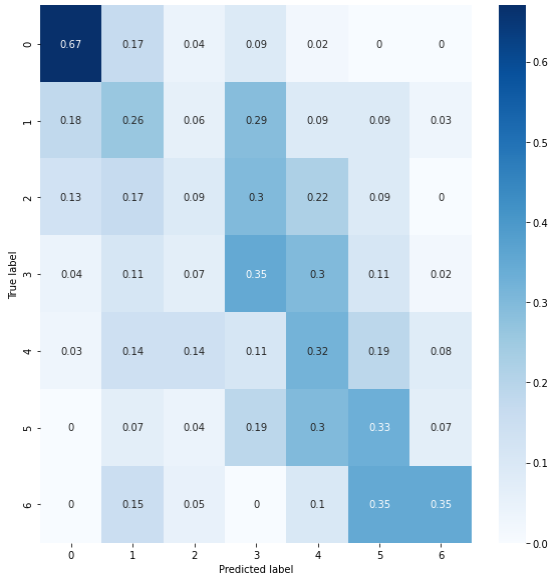

Figure 3.4 Confusion matrix for the baseline model

In conclusion, the accuracy of our baseline model is around 50%. To improve baseline mode, we tried two following methods: Binary Thresholding and Siamese network.

Method 1: Image Subtraction and Binary Threshold

Model Construction and Results

Instead of using all three channels of the original images, we came up with a new idea of extracting the hyperpigmentation area of the face images. Our objective is to extract regions of interest (ROI), that is, the mottled hyperpigmentation area on the face. This enabled the convolutional neural network (CNN) to learn more about the detailed region of interest instead of face contour, fine lines, and wrinkles.

The idea is inspired by *Automatic Acne Detection for Medical Treatment.* Our proposed procedure is as follows:

1. Image Subtraction
    1) Convert RGB image to Grayscale color space to simplify the calculation.
    2) Convert RGB image to HSV color space and extract brightness value (V) from HSV image
    3) Subtract grayscale value out of normalized brightness value to reveal a region of interest.
    4) Multiply the binary image with the brightness layer of the HSV to get the brightness intensity binary threshold image
2. Binary Thresholding to obtain a binary image
    1) Set the threshold value as 0.165 in this case for the best visual result. Any region above 0.165 is considered to be the region of interest, and any region below 0.165 is ignored.

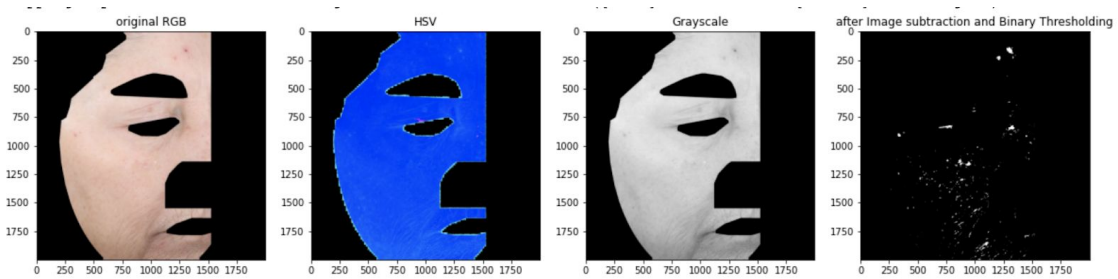

Figure 3.5 Comparison of an image in original RGB, HSV, Grayscale, and Binary Thresholding

2) We took a deeper look at another two images with grade 3 and grade 6 respectively, using binary threshold value 0.165. The image of grade 6 in the original RGB color space shows a larger and darker area of mottled hyperpigmentation. The images after subtraction and binary thresholding show a similar pattern. The binary image with grade 6 has more white pixels than the one with grade 3. Also, its white spots are more widely distributed.

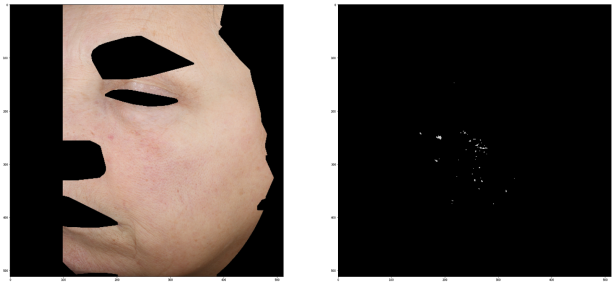

Figure 3.6 An image of grade 3 in original RGB and Binary Thresholding

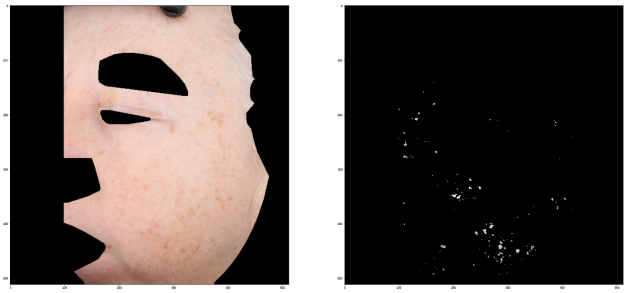

Figure 3.7 An image of grade 6 in original RGB and Binary Thresholding

According to the output images, we observed that irrelevant features, such as face shape, wrinkles are largely removed from our training data. At the same time, the density of the ROI is also well represented by the lightness of the white regions in the Binary Thresholding image since we multiply the normalized brightness layer of the HSV.

In the next step, we used the processed image to train neural networks. We applied the same data preprocessing steps and used the same model as our baseline model: *ResNet50* Transfer Learning, with one drop-out layer, one global average pooling layer, and one dense layer. The model structure and parameter settings are similar to our baseline model. The only difference is that we trained 100 epochs here. The results are as follows.

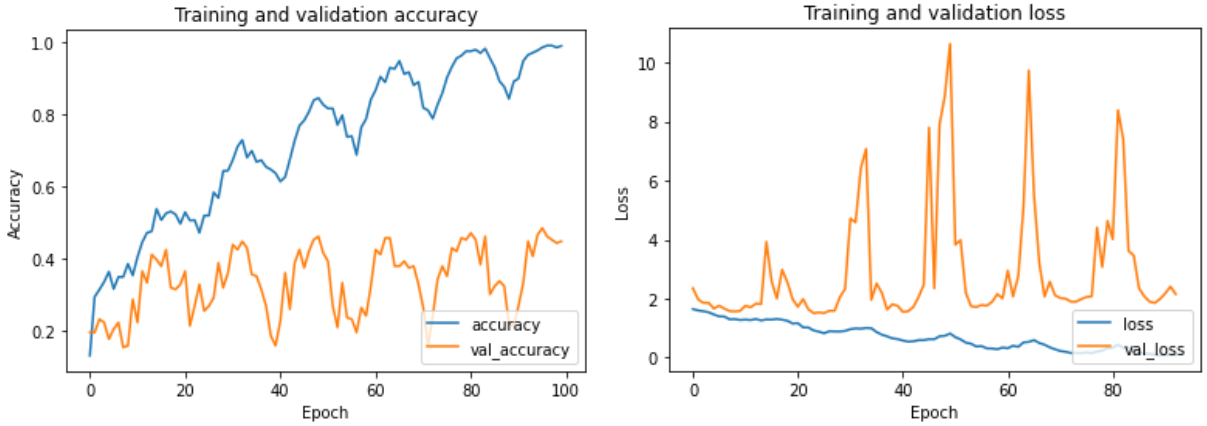

Figure 3.8 Training and validation accuracy and loss for *ResNet50* with Binary Thresholding

```
loss, accuracy = model.evaluate(test_ds)
print("Accuracy: {:2.2%}".format(accuracy))
```

```
8/8 [==============================] - 1s 146ms/step - loss: 2.3987 - accuracy: 0.4232
Accuracy: 42.32%
```

Figure 3.9 Test accuracy for *ResNet50* with Binary Thresholding

Figure 3.8 shows a cyclical pattern for both results of accuracy and loss. This is because we used an exponential cyclical learning rate. The highest validation accuracy is around 45% which does not show an improvement for our baseline model. The validation accuracy is even a little bit lower than that of the baseline model. The test accuracy is 42.32% which is also lower than that of our baseline model.

Potential Problems

After applying the binary threshold method, the model performance has not improved. There are two main possible reasons described as follows.

1. Regions of Interest (ROI) Extraction
   It is likely that the binary threshold method does not capture all regions of interest or exclude all irrelevant features. For example, our method may capture rosacea (a common facial skin condition that causes redness and visible blood vessels) as a region of interest, but the grade does not depend on facial rosacea. Another possible example is that the binary threshold method may capture shadow areas around eye sockets that are not mottled hyperpigmentation areas.

2. Grading Bias
   There are two kinds of bias in our original dataset of grading. First, there exists a bias in images with the same grades due to the difference among graders. Also, since there are masked parts in the images due to the privacy issue, any important information in those parts can also be masked out. Second, there exists a bias between real people's faces and photo images. Graders may give different grades when people are in front of them and when they grade a photo image. It is because there may exist some shadow areas in photos that are caused by face contours. When graders grade an image with some shadow areas, they usually choose to ignore that area whereas our binary threshold method will not exclude the areas with severe shadows.

Method 2: Siamese Network

In order to solve the data lacking problem, we tried a new network called Siamese Network. A Siamese Neural Network is a class of neural network architectures that contains two or more **identical** subnetworks. '**Identical**' in this case is defined as having the same configuration with the same parameters and weights. Parameter updating is mirrored across both sub-networks. Siamese Neural Networks can find the similarity of the inputs by comparing its feature vectors. So even with a few images, we could get better predictions. For every single image, the network compares it with all other images from data sources, and learns their similarity. For example, if we have 100 images, for every single image, the network generates 99 other comparisons, which incredibly increases the training size.

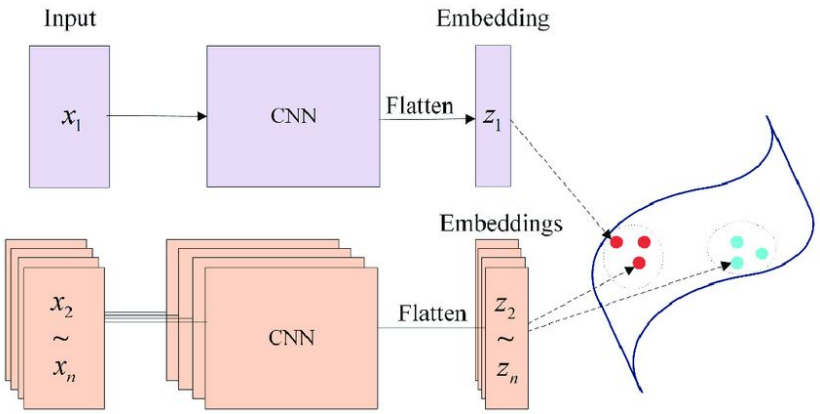

Figure 3.10 Siamese Network

Data preparation

Due to the structure of the Siamese network, which needs two input images at one time, we needed to build appropriate datasets to fit it. In order to do so, we created two identical tensor datasets from a training set with caching and prefetching to maximize efficiency. We applied the shuffle to both of them and used a detect function to check whether their labels are identical or not. Then we fed both of them into the Siamese network.

Siamese network structure

The network itself, defined in the Net class, is a Siamese convolutional neural network consisting of 2 identical subnetworks. Here we used our baseline model using transfer learning with *ResNet50*. After passing through the convolutional layers, we let the network build a one-dimensional descriptor of each input by flattening the features and passing them through a linear layer with 224 output features. Note that the layers in the two subnetworks share the same weights. This allows the network to learn meaningful descriptors for each input and makes the output symmetrical (the ordering of the input should be irrelevant to our goal).

The crucial step of the whole procedure is the next one: we calculated the Euclidean distance of the feature vectors. Usually, the Siamese network performs binary classification at the output, classifying if the inputs are of the same class or not. Hereby, different loss functions may be used during training. One of the most popular loss functions is the binary cross-entropy loss. This loss can be calculated as

$$L = -y \log p + (1 - y) \log(1 - p)$$

where L is the loss function, y is the class label (0 or 1) and p is the prediction.

In principle, to train the network, we used binary cross-entropy loss. Therefore, we attached one more linear layer with 2 output features (equal number, different number) to the network to obtain the logits.

Basic Algorithm

1. We utilized two images (*Image1* and *Image2*). Both of the images are fed to a single Convolutional Neural Network (CNN).
2. The last layer of the CNN produces a fixed-size vector. Since two images are fed, we got two *embeddings* (*h1* and *h2*).
3. The Euclidean distance between the vectors was calculated. We used the *Lambda layer* to calculate the absolute distance between the vectors.
4. The values then passed through a sigmoid function and a similarity score was produced. Since our output is binary in nature, we used the *tf.keras.losses.binary_crossentropy()* function.

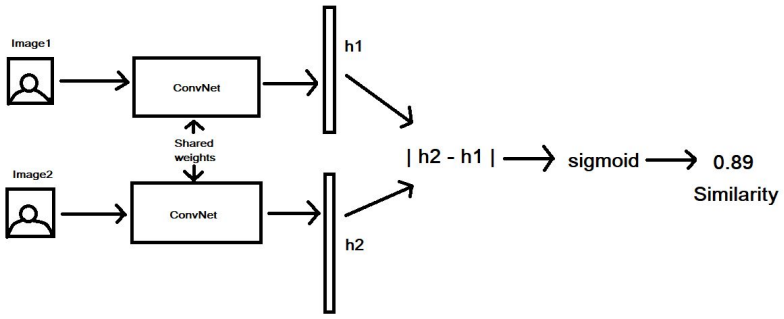

Figure 3.11 Siamese Network Modeling Structure

Sample code:

```
def create_base_model():
    conv_base = ResNet50(include_top = False, weights = 'imagenet',
                         input_shape = (224, 224, 3))


    #conv_base.trainable = False
    x = conv_base.output
    x = tf.keras.layers.Dropout(0.5)(x)
    embedding = GlobalAveragePooling2D()(x)
    embedding = Dense(128)(embedding)
    return Model(conv_base.input, embedding)

def SiameseNetwork(base_model):
    """
    Create the siamese model structure using the supplied base and head model.
    """
    input_a = Input(shape=(224, 224, 3),name = "image1")
    input_b = Input(shape=(224, 224, 3),name = "image2")

    processed_a = base_model(input_a)
    processed_b = base_model(input_b)

    distance_euclid = Lambda( lambda tensors : K.abs( tensors[0] - tensors[1] ))( [processed_a , processed_b] )
    outputs = Dense( 1 , activation='sigmoid') ( distance_euclid )
    return Model([processed_a, processed_b], outputs)
```

Since the output of the Siamese network is a similarity score of two images we fed in, another function is required to do the prediction of the classification of the input image X. Here is how we constructed the classification problem.

To perform actual predictions on images

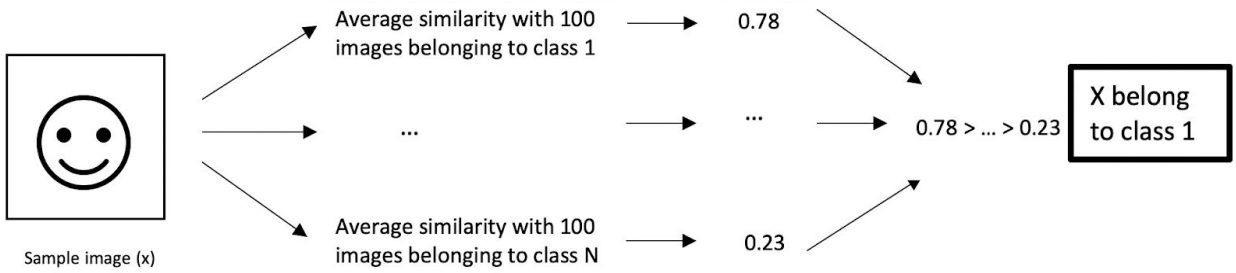

Figure 3.12 Siamese Network prediction structure

1. Group images from the data set based on their class
2. Calculate the similarities between the sample image and images for each class
3. Find the class which has the highest average similarity with the sample image
4. The class with the highest similarity will be the class prediction assigned to the sample image.

Sample code:

```
def calc_similarity(test_paths,test_labels,df):
  path_test = test_paths[0:1]*100
  label_test = test_labels[0:1]*100

  res = []
  for i in set(df.label):
    paths_train = df[df.label == i].path[0:100]
    labels_train = df[df.label == i].label[0:100]

    temp_ds = create_ds_prediction(path_test,label_test,paths_train,labels_train)
    temp_ds = temp_ds.map(lambda image_label1,image_label2:
        (image_label1[0],image_label2[0],image_label1[1],image_label2[1]),num_parallel_calls=AUTOTUNE)
    temp_ds = temp_ds.map(lambda image1,image2,label1,label2 :
        (normalize(tf.cast(image1,tf.float32)),normalize(tf.cast(image2,tf.float32)),label1,label2),num_parallel_calls=AUTOTUNE)
    temp_ds = temp_ds.map(lambda image1,image2,label1,label2 :((image1,image2),detect(label1,label2)),num_parallel_calls=AUTOTUNE)
    temp_ds = temp_ds.batch(BATCH_SIZE).prefetch(buffer_size=AUTOTUNE)
    prediction = new_model.predict(temp_ds)
    avg_similarity = sum(prediction)/len(prediction)
    res.append([i,avg_similarity])

  All_res = pd.DataFrame(res, columns = ['class', 'similarity'])
  return All_res

def prediction(similarity_table):
  return similarity_table.max()[0]
```

To train the model we used *Adam* as our optimizer and *Sparse Categorical Cross-Entropy* as our loss function. Different from the baseline model, we used 1e-8 as our learning rate instead of the cyclical learning rate, because we rarely faced the local optimal problem when we trained the Siamese network. We used both the *Accuracy* and *RMSE* as the evaluation metrics.

After running 100 epochs on the Siamese network with batch size 64, we achieved 86.4% training accuracy and 84.1% validation accuracy. Compared to the base model and the binary threshold method, the accuracy gains a great improvement.

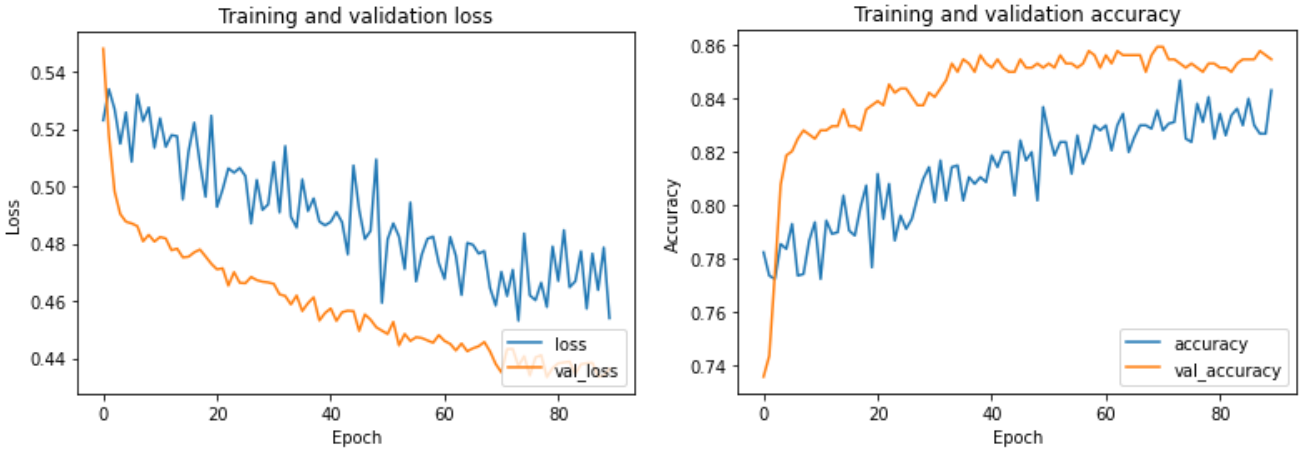

Figure 3.13 Training and validation accuracy and loss for Siamese Network

The following plot shows the Root Mean Square Error for the Siamese network. We could see that the model has a 0.37 RMSE for validation data set, which is close to the target of 0.3 RMSE.

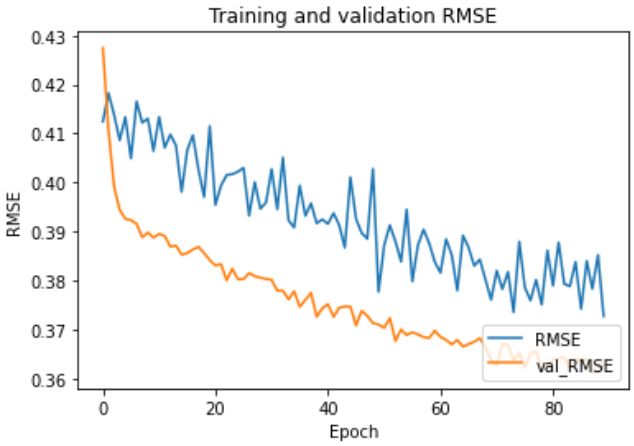

Figure 3.14 Training and validation RMSE for Siamese Network

The test RMSE is 0.39 and the Accuracy is around 81.25%, which is consistent with the validation set.

```
loss, accuracy, RMSE = siamese_network.evaluate(test_ds,steps= 32)
print("Loss: {:1.2}".format(loss))
print("RMSE: {:1.2}".format (RMSE))
print("Accuracy: {:2.2%}".format(accuracy))
```

```
32/32 [==============================] - 6s 195ms/step - loss: 0.4804 - accuracy: 0.8125 - root_mean_squared_error: 0.3900
Loss: 0.48
RMSE: 0.39
Accuracy: 81.25%
```

Figure 3.15 Test accuracy for Siamese Network

Potential improvements

The Siamese network performed very well on the current dataset, but there are still two factors that could be improved.

1. As we previously discussed, in order to do the actual classification, we need to calculate the similarity of the input image x and images from each class in our dataset. In our model, we chose 100 images from each class and calculated the average similarity. More specifically, the model processes image x and one image from the training set separately through our baseline CNN model to get the embeddings of them and needs to repeat this process 100*N times, where N states for the number of classes. When another image y comes in, we need to do all the process again. The calculation process takes a long time. In order to make the model more efficient, we can do the precalculation of the embeddings of all images from our training dataset and make the average vectors of embedding for images in each class be prepared. Therefore, when a new image needs to be predicted, we can only process this single image through our baseline model and calculate the embedding of it. It will make the classification process much faster.

2. For the identical subnetworks, we used our baseline model for the embedding and did get a good result, but the baseline model is a pre-trained model of transfer learning, so it is quite complicated and will take a long time to train. However the role of the identical subnetworks is only to do the embedding and does not have to learn all the detailed features of images. Therefore we can build a CNN model from scratch with a simple structure, which could also perform well but be more efficient and save computing power.

4 Conclusion

In conclusion, we developed an automatic grader for grading mottled hyperpigmentation from facial images taken in clinical studies. To achieve this goal, we attempted several deep learning models. As a result, we employed three models:

1. Baseline Model: a transfer learning model with *ResNet50*
2. Binary thresholding as image preprocessing in the baseline model
3. Siamese Neural Network model with the baseline model as subnetworks

We chose the Siamese Neural Network model to be our final model because the Siamese Neural Network model outperformed the other two models and achieved test accuracy as high as 81.25% and RMSE (root mean squared error) as low as 0.39.

In the future, Unilever could follow the proposed future work in the next section.

5 Future Work

Collect More Data With Extreme Grades

Because of the lack of data, we grouped the data with extremely low and high grades into two groups in order to make the data more uniformly distributed. However, these methods result in lack of precision when predicting cases of extreme grades. If it is applicable, we would recommend Unilever to collect more data in grades lower than 3, and grades greater than 5. Hence, the model could have precise predictions for all grades.

Improve the Baseline Model with Other Transfer Learning Model

We used *ResNet50* as our transfer learning model. Throughout our research, we found that other pre-trained models such as *ResNet152 ImageNet Caffe* may have better performance. In the research paper *Assessing the Severity of Acne via Cell Phone Selfie Image Using A Deep Learning Model* by Hang Zhang, they used *ResNet152 ImageNet Caffe* and achieved Root Mean Square Error of 0.482 which is better than the target performance bar 0.517 which was set by *Nestle Skin health and Microsoft* as the highest Root Mean Square Error among 11 dermatologists. We would recommend Unilever to try to train this model and compare the result with *ResNet50*.

Crop Facial Image to Extract Regions of Interest More Precisely

We would recommend Unilever to try to extract skin patches from facial images. From the forehead, both cheeks and chin of each face image by applying the pre-trained *Shape Predictor 68 Face Landmarks* (landmark model) published by Akshayubhat on github.com. The next step is to calculate the weighted average score of the whole face.

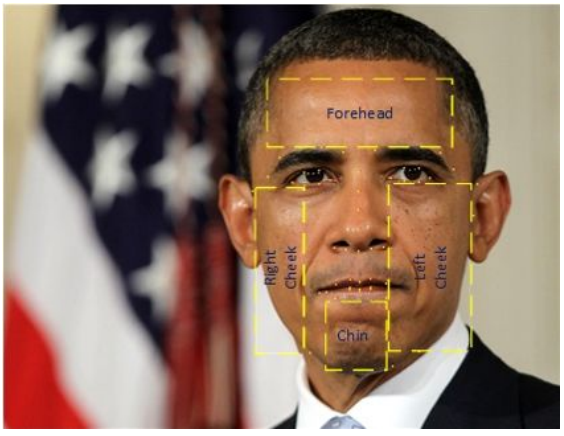

Figure 5.1 Illustration of extracting skin patches

References

[1] T. Chantharaphaichi, B. Uyyanonvara, C. Sinthanayothin and A. Nishihara, "Automatic acne detection for medical treatment," 2015 6th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES), Hua-Hin, 2015, pp. 1-6, doi: 10.1109/ICTEmSys.2015.7110813
http://www.cephsmilev2.com/chanjira/Papers/2015/ICICTES2015Thanapa.pdf

[2] T.Chantharaphaichit, Thanapha, B. Uyyanonvara, C. Sinthanayothin and A. Nishihara. “AUTOMATIC ACNE DETECTION WITH FEATURED BAYESIAN CLASSIFIER FOR MEDICAL TREATMENT.” (2015).
http://www2.siit.tu.ac.th/bunyarit/publications/2015_RIIT2015_BKK_Khim.pdf

[3] Will Koehrsen, “Neural Network Embeddings Explained”
https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

[4] Hang Zhang, “Assessing the Severity of Acne via Cell Phone Selfie Images Using A Deep Learning Model”
https://devblogs.microsoft.com/cse/2019/02/05/assessing-the-severity-of-acne-via-cell-phone-selfie-images-using-a-deep-learning-model/

[5] Sean benhur J, “A friendly introduction to Siamese Networks”
https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942

[6] Akshayubhat, “81 Facial Landmarks Shape Predictor”
https://github.com/codeniko/shape_predictor_81_face_landmarks