hAIr: A Hairstyle Recommender System Using Machine Learning

Sark Xing
Technical University of Eindhoven
p.xing@student.tue.nl

Ward de Groot
Technical University of Eindhoven
w.d.groot.1@student.tue.nl

Yizhou Liu
Technical University of Eindhoven
y.liu10@student.tue.nl

Lara Leijtens
Technical University of Eindhoven
l.l.leijtens@student.tue.nl

Abstract
Choosing a new hairstyle can be a difficult, impactful decision. Especially envisioning if a haircut would suit the individual is hard. With the analysis responses from facial recognition APIs and supervised machine learning, a relation between facial features and hairstyle is ought to be found in this project, so that a hairstyle recommender system, called "hAIr", can be created. The system recommends hairstyles that suit the individual's characteristics. This is based on a neural network learning algorithm, which is trained with features, extracted from 1060 images of people, relating to 53 different hairstyles. The trained network reaches an accuracy of 28.10% when validated with images that were not used for training. This can be improved by trying different combinations of input variables, or using a different conversion for the values that were gained from the APIs. It is also possible that the APIs are not completely accurate. A third possibility for improvement would be to use a different learning algorithm, such as k-Nearest Neighbors or naive Bayes.

Introduction

Context

Going to the hairdresser can come with difficult decision making. A new haircut is a decision that cannot be changed for at least six weeks. So, for this amount of time it is visible to everyone that a wrong decision has been made. Especially for occasions where a good haircut is essential, such as a job interview, the wrong haircut can badly influence a first impression. Not only the impact of the decision makes it hard to choose, also the large amount of options causes difficulty. There are so many different hairstyles that it can seem like an impossible decision to make.

At a barbershop or hairdresser there are often magazines provided with examples of haircuts. However, these examples are not personalized. So when it is boiled down to a selection that fits the client's style, it is not guaranteed that the haircuts will look good on the client. This is because a haircut can look different on different people. An experienced barber or hairdresser could give advice on this, but they are hard to find and often expensive.

Challenge

The challenge that is ought to be tackled here is to make it easier for people to decide on a new hairstyle by minimizing the risk and limiting the options. The system is meant for both men and women of all ages that want to try something new, but do not know what haircut would suit their facial features and do not have an experienced hairdresser who can advise them on the matter. Hairstyle does not only correlate with facial features, but also with other elements, such as a person's clothing style or a person's current hairstyle. However, this is often something that people do know about themselves, once they are presented with an option. To decide if they like the style or not can be assessed by looking at another person that has that haircut. However, whether it will suit them personally is hard to assess, so this is the challenge that is aimed at.

Competition

Currently there are several products in development that are similar to the envisioned product for this project. However, not one of them is dominating the market, so there is still room for competitors.

Face It [18] is an application that detects the shape of person's face. Hereafter, the user has to provide personal information, such as occupation. Based on these combined input features, a recommendation for hairstyle, including facial hair, is given. This differs from the envisioned product, because the user does not need to provide any information other than a picture. This increases the ease of use. However, Face It has some additional features that could be added in later development of the envisioned product, such as a list of nearby hairdressers and recommended hair products.

The app Hair Color of ModiFace [14] focusses on hair color change. It gives a real time visualization of a different hair color mapped on a person's hair through live camera input. This could be interesting for collaboration, since the product of this project does not focus on hair color, but on hair style.

Both Dreambit [5] and FaceApp [7] have a functionality to map different hairstyles on images of the user's face. This is not the area that is currently focused on, but the technology would be interesting to apply in later development.

Approach

Envisioned solution

To complete the challenge, the envisioned solution is an app that provides users with a limited amount of hairstyles which will all suit the user's face. This will minimize the risk of choosing a hairstyle that does not look good and makes it easier to choose. It is expected that it can be determined whether a hairstyle will look good or not on a person, based on facial features and characteristics such as age and gender. However, this relation is not easily grasped in a formula or logic rule. Therefore, machine learning is used to find matches. More specifically, supervised machine learning, because hairstyles have existing names and do not need to be categorized by the machine.

Learning algorithm

As learning algorithm for the system, a Multi Layered Perceptron (MLP) is chosen, which is widely used in many similar recommender cases to solve realistic problems [11][17]. An MLP is a type of Neural Network (NN) that deals with non-linearly separable data. An artificial neural network is a learning algorithm consisting of nodes and weighted connections. The nodes are grouped in layers. For an MLP there is one input layer that contains the input variables, one or multiple hidden layers and one output layer that contains the output targets.

Each node from one layer is connected to all nodes in the next layer. Training is done by using Neuroph [15], a lightweight Java neural network framework. During the training, the network looks for the optimal combination of weights. This works as following. The network takes one person's input variables and target and guesses what the weights could be. Then, it checks if this has led to the correct target (supervised learning). Depending if this is correct or not, the network adjusts the weights of its next guess. The delta rule is applied so that the amount of adjustment is gradually descending. The delta rule is a special case of the more general backpropagation algorithm that was used for this project.

Additionally, other than using a NN, the k-Nearest Neighborhood (kNN) algorithm could also be fitting for this problem. The kNN classification maps every input in multidimensional space. The position is determined by the input features, in this case the shape of the face or the skin color. This position is labeled with the target, in this case the hairstyle. When a user provides new input features, a position is again determined based on their values. Since there is no label provided, the system will check which point or points are closest to it. The amount of points that are taken into account is called the k-value, and it is usually a small integer number. So, basically the system finds the person that the user looks like most and advises the same hairstyle as this person.

Furthermore, naive Bayes could be used as a classifier as well. It uses a probabilistic model to check independence between features and is based on Bayes' theorem. A simplified dependence applied to this case would be: what is the chance of a bobline hairstyle when a person has an oval face? This can be calculated by multiplying the chance on a bobline hairstyle by the chance on an oval face when someone has a bobline hairstyle, and dividing the answer by the chance that a person has an oval face with any hairstyle. Repeating this for all possible independencies, and combinations, e.g. the chance on a bobline when a person has an oval face ánd a pointy nose, results in a prediction model.

However, amongst the researchers there is no experience with these two latter algorithms. Therefore, it would take a lot of time to find out how the three algorithms work and design a test to compare the three algorithms. Hence, a pilot test for the NN was conducted to investigate whether this learning algorithm would be fitted for the task at hand.

Data acquisition

To train a supervised AI, it must be fed with correct combinations of input as well as output variables. The input variables are facial features and the output variables, the targets, are varying hairstyles. For this, there are several databases available. Figaro 1k [20], a hairstyle dataset, consists of seven hairstyles (straight, wavy, curly, kinky, braids, dreadlocks and short) with 150 images for each hairstyle. For Figaro 1k, it includes only seven haircuts, which is not sufficient for training a Neural Network and too general to be recommended for customers. The human face and hair dataset from Open Images V4 [16] (powered by Google AI) can provide abundant and copyright-free images in the natural settings. However, it does not provide hairstyle labels. The data used for this system is derived from the Hairstyle30k database [21], which is the largest dataset regarding the number of hairstyle classes within the community. It contains 12,076 images, which are distributed under 64 labels. The hairstyle classification method developed by Hairstyle30k was based on people attributes and face attributes by using 1) hand-crafted visual features, such as SIFT [12] and LBP [19], 2) the recent deep features and 3) multi-task methods for learning facial attributes. According to the attributes, the researchers of Hairstyle30k manually coined and annotated 64 labels for different hairstyles.

Since all these images are labelled with the target data (hairstyle), it is practical to use the same images as input. However, the exact input variables need to be specified. The aim is to relate facial features to the hairstyles, so the facial features of the people in the 30k database need to be collected. To achieve this, there are several options available: 1) looking at the images and noting the features by hand, either done by the researchers or by experts (hairdressers), 2) creating a machine learning algorithm that extracts the features from the images, and 3) using an existing API that extracts the features. The third option was selected because option 1 and 2 are time-consuming. Furthermore, based on the limited experience of the researchers, 1 and 2 would not guarantee good results when performed by the researchers.

Moreover, when new input is presented by the user, it would be easier to upload a photo than to write down all your personal facial features. Since the average user does not have the experience. Thus, the input data to train the system is gathered in the same way the data is gathered in the use setting.

API SELECTION

As explained above, the input data was extracted from the images of the hairstyle database using an API. BetaFace API [2], an open API for face recognition, was found initially and seemed perfectly fitted for the goal. However, the accuracy on several elements, such as age, was not very high. Therefore, a second search for APIs was conducted.

There are several face recognition APIs available. All have their own specialties and limitations. To select which one was best fitting for this case, multiple APIs were considered: Lambda Face Recognition API [10], FaceX [8], Animetrics [1], Azure Face API [13], IBM visual recognition [9], Face++ [6], and DeepFace [3]. Every API was evaluated on seven available input options: gender, ethnicity, face characteristics, face landmarks, facial hair, glasses and age. These seven were all believed to be in relation to the hairstyle of a person. In table 1 can be viewed which categories were enabled by which API. Azure Face API scores best, since it includes most of the features.

However, not one of the considered APIs was able to distinguish face characteristics, such as face shape. Since it is the core element in the relation between face and hairstyle, it was chosen to use a combination of Azure Face API and BetaFace API. The Azure Face API can detect faces in images and extract face-related attributes needed for our recommender system. The attributes include gender, age, facial hair and glasses. The API of BetaFace is used to retrieve information on the face shapes detected in the dataset's images.
Combining these two API's, enough relevant information of the images is extracted to make a recommendation.

Input variables

The features that were selected as input variables can be viewed in table 2. These were determined based on the following reasons. General features, being gender and age, were included because they are expected to deliver a common ground, as hairstyles are usually fitting for either men or women and associated with a certain age. Facial hair was also included, since this should match the hairstyle. As facial hair was not part of the hairstyle categories in the database, it is not part of the output recommendation. Glasses and make-up are features that could change on a day-to-day basis. However, it is assumed that people who wear glasses or make-up do so consistently, and therefore these are used as input variables. All other facial features are features that are consistent for the user and that cannot be changed easily.

There were more features extracted from the database's images through the APIs. Most of them were either unrelated (e.g. blurry image), temporary

Features	Lambda	FaceX	Animetrics	Azure	IBM	Face++	DeepFace
Gender	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Ethnicity					🟩	🟩	🟩
Face characteristics							
Face landmarks		🟩	🟩	🟩		🟩	🟩
Facial hair				🟩			
Glasses				🟩			
Age				🟩	🟩		🟩

Table 1. API comparison based on seven features.

(clothing color), or on a non-dichotomous nominal scale that cannot be mapped trustworthy (e.g. race with a value of Asian-middle-eastern, Asian, African American, Hispanic, white, middle eastern, or other). A list of all possible features that could be extracted through Betaface and through Azure can be found in Appendix 1 and 2, respectively. These lists include the corresponding values.

#	Element	Value	Mapping	API
1	Gender	male/female	1, 0	Azure
2	Age	0-100	age/100.0	Azure
3	Moustache	0-1	-	Azure
4	Beard	0-1	-	Azure
5	Sideburns	0-1	-	Azure
6	Glasses	No glasses, Reading glasses, Sunglasses	0, 0.5, 1	Azure
7	Oval face	yes/no	Boolean if yes: 0.5 + confidence/2 if no: 0.5 - confidence/2	BetaFace
8	5 o'clock shadow	yes/no		BetaFace
9	Arched eyebrows	yes/no		BetaFace
10	Bags under eyes	yes/no		BetaFace
11	Bangs	yes/no		BetaFace
12	Beard	yes/no		BetaFace
13	Big lips	yes/no		BetaFace
14	Big nose	yes/no		BetaFace
15	Bushy eyebrows	yes/no		BetaFace
16	Double chin	yes/no		BetaFace
17	Heavy make-up	yes/no		BetaFace
18	High cheekbones	yes/no		BetaFace
19	Narrow eyes	yes/no		BetaFace
20	Pale skin	yes/no		BetaFace
21	Straight hair	yes/no		BetaFace
22	Wavy hair	yes/no		BetaFace
23	Chin size	extra small, small, average, larger, extra large	0, 0.25, 0.5, 0.75, 1	BetaFace
24	Eye distance	extra close, close, average, far, extra far	0, 0.25, 0.5, 0.75, 1	BetaFace
25	Head width	extra narrow, narrow, average, wide, extra wide	0, 0.25, 0.5, 0.75, 1	BetaFace
26	Mouth width	extra small, small, average, larger, extra large	0, 0.25, 0.5, 0.75, 1	BetaFace

Table 2. All 26 features that were selected for training the NN, including their corresponding values, the mapping method and the API that was used to extract the features.

It can be discussed whether wavy hair and straight hair as input features are chosen well, since they describe the hairstyle, which is the targeted outcome.

Process

Pilot test

To see if the NN would be able to recognize hairstyles based on face features, a pilot test was designed. In case it would give positive results, this type of algorithm would be used for the final system as well. If it would give negative results, the two other options, kNN and naive Bayes, would be explored.

Figure 1. Examples of Sam (top row) with hairstyle 1, 2 and 3 and Harry (bottom row) with hairstyle 1, 4 and 5.

For this test, two different people, Harry and Sam, were selected. The features that were taken into account were the roundness of their head, the whiteness of their skin, whether they have glasses or not and whether they have a beard or not. Both Harry and Sam had three possible hairstyles, one of which was the same for both. This can be viewed in figure 1.

Hundred Sams and Hundred Harrys were generated, following the intervals shown in table 3. These 200 'people' were used to train an MLP in Neuroph. After the system was trained, the result was validated by using newly generated Harrys and Sams. This showed that the system could always recognize whether the person was a Sam or a Harry. The error rate of the training was 0.03, and in the validation with 10 different Sam and Harry, the network proposed a fitting hairstyle.

Person	Roundness	Whiteness	Glasses	Beard
Sam	Random (0.75 – 0.95)	Random (0.75 – 0.95)	0	1
Harry	Random (0.25 – 0.45)	Random (0.75 – 0.95)	1	0

Table 3. Features for 100 artificially generated Sams and 100 artificially generated Harrys.

This showed that an MLP is able to categorize faces and find a fitting hairstyle. Therefore, no more time was wasted on trying out different algorithms, such as kNN or naive Bayes, but all time was invested in developing a system using this learning algorithm.

Data filtering

The data provided 64 different targets (hairstyles). However, not all of these were fitting for the method used in this project. The following hairstyles were eliminated, because there were less than 20 images that could be recognized by both APIs. In most cases this was because the pictures were taken from the back of the head, since the hairstyle was only visible from the back. Therefore, no facial features could be seen. Following this reasoning, 11 hairstyles were eliminated: long hair, devilock, ducktail, horseshoe flattop, rattail, French twist, tonsure hair, quiff, fontange, Jeong-eun Kim hair, and khokhol hair. Elimination of these hairstyles resulted in 53 usable hairstyles.

Of each hairstyle, the first 20 images were selected for training the network. Pictures with more than one recognized face were deleted from the training data, because the hair style is only determined for one of the faces. These were replaced by the next picture (number 21) in the database.

Neural network experiments

The 26 inputs gathered from both API's, combined with the filtered output class data of 53 outputs, formed the base of the MLP neural network. The amount of hidden layers and their amount of nodes still needed to be defined. Therefore, the team explored which neural network structure resulted in the most accurate and confident outcomes. Having more than one hidden layer within an MLP network does not influence these outcomes [4]. That is why having multiple hidden layers was eliminated. Then, the amount of nodes in the hidden layer was experimented with, as well as with the learning rule and amount of iterations. Within Neuroph Studio, all the different combinations were

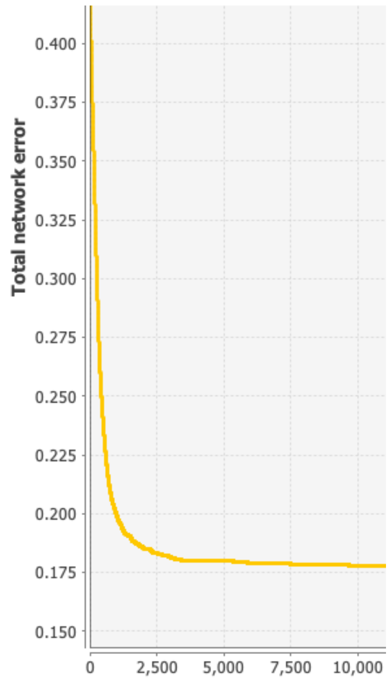


Figure 2. Total network error graph of NN training with 50 nodes and backpropagation as learning rule, showing results for the first 10.000 out of 50.000 iterations. The graph continued descending slowly until an error of 0.157 was reached.

tried out. The nodes of the hidden layer varied between 5, 10, 15, 20, 35, 40, 50, 75 to 100 nodes, all using different learning rules. To prevent the outputs to be 0, a bias rule was applied within the inputs.

Training
The amount of iterations was set to 50,000. Each NN used the same dataset of 20 images per class (1060 images in total), to train the network. A learning rate of 0.2 and a momentum of 0.7 was applied for each NN. These experiments showed that having 50 nodes in the hidden layer while using backpropagation as learning rule gave the lowest error rate (0.157), see figure 2.

Validation
After all the networks completed training for 50,000 iterations, they were all validated using two different methods. One was validating the systems by inputting the used images, to see if the system was correctly trained. The other was utilizing three images of all 53 hairstyles which were not used to train the system. In this way, the accuracy of the system can be measured. The validation of the network with our highest accuracy rate had a total network error of 0.18. This network uses 22 input nodes, 50 nodes in the hidden layer, and 53 output nodes.

In order to validate whether the system had been trained correctly, the same images used to train the network were put into the system again. This method showed an accuracy rate of 64.60% (table 4).

Putting in unused, labeled images of the same dataset which was used to train the network showed a recommendation accuracy rate of 28.10%. The recommendation shows the three best fitting results, according to the image that has been put in table 5.

Training validation results

No. of recommendations	Accuracy rate (%)
1	64.60
3	65.10
5	66.30
10	73.00

Table 4. Accuracy rate related to number of recommendations by validation with trained data.

Trained network validation results

No. of recommendations	Accuracy rate (%)
3	28.10
5	41.20
10	58.80

Table 5. Accuracy rate related to number of recommendations by validation with fresh (validation) data.

As seen in both tables 4 and 5, the accuracy of the system increases when giving more recommendations. This is a logical outcome, since giving more recommendations would also imply having a higher chance that the right hairstyle recommendation is included as output.

Results
After all the training and validation tests, one final neural network which has the highest accuracy rate was chosen for the recommender system ‘hAIr’. This system is explained in this video: https://www.youtube.com/watch?v=H_Gx9_LJBZs&feature=youtu.be. This video shows the working prototype.

Concluding, we can say that our hairstyle recommender system has an accuracy rate of 28.10%. Even though this is not the desired outcome, it can be argued why the system shows these results. These are further discussed in the section Discussion.

Implementation

Barber

The primary implementation area of the system is at a barbershop. A concept video has been made to show the envisioned implementation, as can be viewed here: https://www.youtube.com/watch?v=aNsfy-b7bcY&feature=youtu.be. Note that this video does not show the working prototype.

Wigs

Another field of implementation is to use the system in hospitals or wig stores. Because the system is focused on facial features, recommendations can be provided for people without hair as well.

Discussion

The accuracy of the trained network was 28.10%. This low success rate can be explained or improved by several aspects. These are discussed in the following paragraphs.

Inaccuracy API

As seen in the results section, our hairstyle recommender system does not have a high accuracy levels for recommending the right hairstyle according to one's facial features. We argue that the low accuracy could be caused by the inaccuracies of the API's that were used. The API gives an output, together with a confidence. If the API outputs are of low confidence, it influences the overall recommendations, since the outputs of the API's are used as input for our neural network.

Input conversion

Currently, the input features were converted linearly. However, a different conversion could have led to different results. For example, a Sigmoid function could be applied. The Sigmoid function, or logistic curve, converts a set of integers into a set of decimal numbers between 0 and 1. It would be wise to try this in future development, to see if it improves the accuracy of the network.

Different learning algorithm

In hindsight, it seems that kNN or naive Bayes would have been better fitting to the problem. However, this is still speculation. For future development, it is important these options are explored, since the NN does not currently reach the desired accuracy. The downside of kNN is that the training time will increase every time new data is added. This is only the case when a feedback loop is used.

Future work

To improve the implementation and relevance of the system, several ideas are suggested for further development, as can be read underneath.

Unfitting recommendations

Currently, the system does not consider a person's current hairstyle, so it is possible that a recommendation is given that is not fitting at the moment. For example, a hairstyle that requires longer hair than the person currently has. This can still be interesting to know, since the user can decide to wait until his/her hair has grown longer. There are other possible mismatches, such as a difference in color or hair type (straight or curls). This can be changed, by coloring the hair or getting permanent curls or straightening. However, this is more expensive than a regular haircut.

There are several ways in which this can be solved. The system's concept can be changed from a recommender to a matchmaking service, so that the user can select preferred hairstyles, after which the system checks if the provided hairstyles match the facial features. Another possibility is that the user can answer questions about the preferred price of the hairstyle.

Style
The system is set up in such a way that it is very easy and fast to use, because only one image is needed to receive a recommendation. Also, three options are recommended out of 53 options. This combination of choices leads to a recommendation for the average person. Therefore, the system will not work for people with a distinctive style. To include those people in the target group, the style of the user can be gained from user input apart from the image. This input is used in addition to the facial features. Another option would be to let users select the hairstyles they are interested in and thereafter let the system recommend which is best fitting with their features. Moreover, additional hairstyles can be added to increase flexibility.

Trends
The system as it is designed right now does not take into account trends in hairstyles. However, it would be interesting to look at how the system could develop over time. Whenever our system is implemented at hairdressers, new data can be fed to the system, which can influence the overall outputs. Whenever for example a customer wants a hairstyle that has not been defined yet by the labels in the dataset, a new label may be created. Real-time updating the system would make our system more connected.

Feedback loop
Currently, the system functions with a NN that is trained prior to use. In the future, when the system is in use, a feedback loop can be set up. Based on the hairstyle users select by clicking "I want this!" in the app, the network can be adjusted based on their facial features in relation to their desired hairstyle. When this is implemented, the neural network needs to be trained again. By doing so, trends in hairstyle will get incorporated as well, since popular hairstyles will be chosen more often.

Current hairstyle
An option is to take into account a person's current haircut and recommend a new hairstyle based on people who changed their hairstyle and previously had the same haircut. Currently, this was not integrated, because data was not available to the researchers. However, when data is gathered (e.g. from barbershops) this might have a positive influence on the accuracy of the model. However, it can often be the case that people who go to the hairdresser want to keep the same hairstyle, because they are still happy with it. Since this is system is only meant for people who want a change, either these data should be left out, or the data should be used in combination with a score of happiness about the current hairstyle. When using the system, users should also give such a score as input, as well as their current hairstyle.

Acknowledgements
This project would not have been possible without the help of other parties. First of all, the hairstyle database was made available to us by Yanwei Fu. Secondly, we would like to thank our teachers and peer students for their critical feedback, which helped us improve the product. Especially, we would like to thank our coach, Erik van der Spek, for his advice. Furthermore, we are thankful for expert input on the topic of Artificial Intelligence from Bart Bussmann. Lastly, we are grateful for the hospitality of Marjan, who let us shoot our video at the hair salon 'For Your Hair'.

References

1. Animetrics: Face recognition. Available from http://animetrics.com/.
2. BetaFace: Open API for face recognition. Available from https://www.betafaceapi.com/.
3. Deepface: Face detection, verification, recognition and emotion analysis APIs. Available from https://deepface.ir/.
4. Doug (2018, July 22). How to choose the number of hidden layers and nodes in a feedforward neural network? Message posted to https://stats.stackexchange.com/q/1097.
5. Dreambit. Retrieved from digitaltrends.com/photography/dreambit-computer-vision.
6. Face++: Cognitive services. Available from https://www.faceplusplus.com/.
7. FaceApp Inc: FaceApp, AI Face Editor. Available from https://itunes.apple.com/us/app/faceapp-ai-face-editor/id1180884341.
8. FaceX: Face recognition API for your apps. Available from https://www.facex.io/.
9. IBM: Watson visual recognition. Available from https://www.ibm.com/watson/services/visual-recognition/.
10. Lambda Labs: Face recognition API. Available from https://lambdalabs.com/face-recognition-api.
11. Sangjae Lee and Joon Yeon Choeh. 2014. Predicting the helpfulness of online reviews using multilayer perceptron neural networks. Expert Systems with Applications41, 6 (2014), 3041–3046.
12. David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision60, 2 (2004), 91–110.
13. Microsoft Azure: Face. Available from https://azure.microsoft.com/en-us/services/cognitive-services/face/.
14. ModiFace: Hair Color. Available from https://itunes.apple.com/nl/app/hair-color/id485420312.
15. Neuroph: Java neural network framework. Available from http://neuroph.sourceforge.net/.
16. Open images dataset V4 + Extensions. Available from https://storage.googleapis.com/openimages/web/index.html.
17. Octavio Salcedo Parra, Gustavo Garcia, and Brayan Reyes. 2014. Traffic forecasting using a multi layer perceptron model. Proceedings of the 10th ACM symposium on QoS and security for wireless and mobile networks - Q2SWinet 14(2014).
18. Pallab Paul. 2017. Face It: The artificially intelligent hairstylist. Intel AI Academy. Retrieved from https://software.intel.com/en-us/articles/face-it-the-artificially-intelligent-hairstylist.
19. Toon De Pessemier, Kris Vanhecke, and Luc Martens. 2016. A scalable, high-performance Algorithm for hybrid job recommendations. Proceedings of the Recommender Systems Challenge on - RecSys Challenge 16(2016).

20. Muhammed Umar Riaz, Michele Svanera, Sergio Benini. 2017. Multi-class hair image database with ground truth: Figaro extension. Available from: http://projects.i-ctm.eu/it/progetto/figaro-1k.
21. Weidong Yin, Yanwei Fu, Yiqing Ma, Yu-Gang Jiang, Tao Xiang, and Xiangyang Xue. 2017. Learning to Generate and Edit Hairstyles. Proceedings of the 2017 ACM on Multimedia Conference - MM 17 (2017).

Appendices

Appendix 1: Possible classifiers and return values

Betaface API

From Betaface API, all possible classifiers with corresponding return values are listed. These values are retrieved from https://www.betafaceapi.com/wpa/index.php/documentation on 2 February 2019.

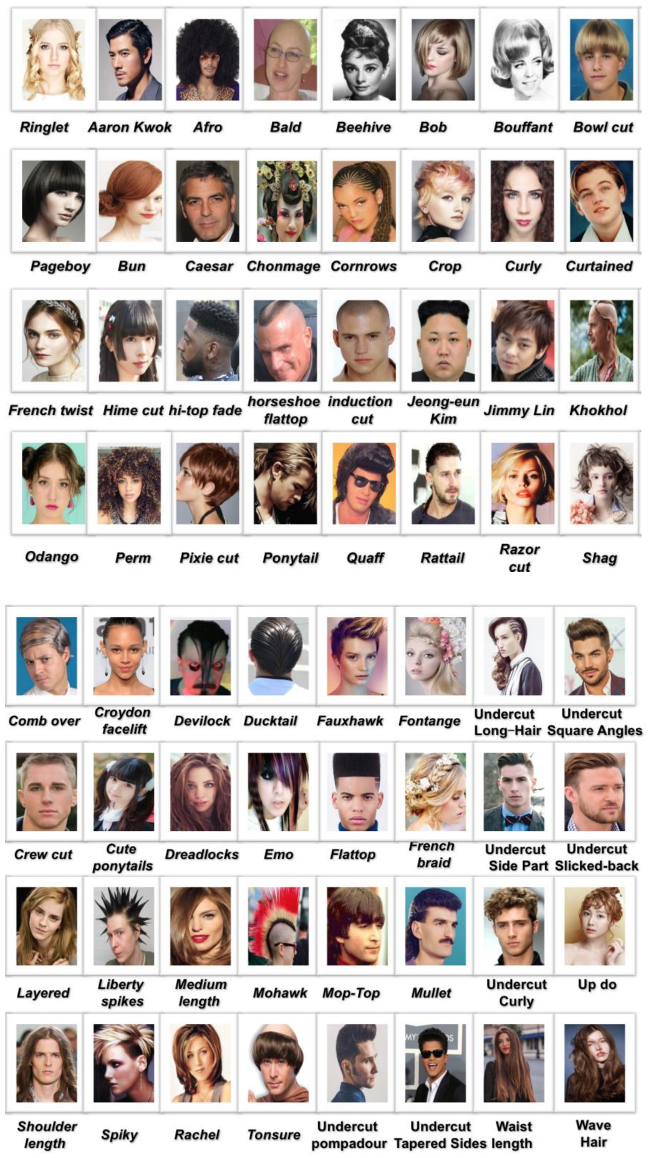
Classifier	Return value
5 o'clock shadow	yes, no
age	approximate age value
arched eyebrows	yes, no
attractive	yes, no
bags under eyes	yes, no
bald	yes, no
bangs	yes, no
beard	yes, no
big lips	yes, no
big nose	yes, no
black hair	yes, no
blond hair	yes, no
blurry	yes, no
brown hair	yes, no
bushy eyebrows	yes, no
chin size	extra large, large, average, small, extra small
chubby	yes, no
color background	RGB hex color value, for example 4f3530
color clothes middle	RGB hex color value
color clothes sides	RGB hex color value
color eyes	RGB hex color value
color hair	RGB hex color value
color mustache	RGB hex color value
color skin	RGB hex color value
double chin	yes, no
eyebrows corners	extra low, low, average, raised, extra raised
eyebrows position	extra low, low, average, high, extra high
eyebrows size	extra thick, thick, average, thin, extra thin
eyes corners	extra low, low, average, raised, extra raised
eyes distance	extra far, far, average, close, extra close
eyes position	extra low, low, average, high, extra high
eyes shape	extra round, round, average, thin, extra thin
gender	male, female
glasses	yes, no

glasses rim	yes, no
goatee	yes, no
gray hair	yes, no
hair beard	none, short, thick
hair color type	black, blond, red, brown, brown light, not natural light, not natural
hair forehead	yes, no
hair length	none, very short, short, average, long, very long
hair mustache	none, short, thick
hair sides	very thin, thin, average, thick
hair top	very short, short, average, thick, very thick
head shape	extra heart, heart, average, rect, extra rect
head width	extra narrow, narrow, average, wide, extra wide
heavy makeup	yes, no
high cheekbones	yes, no
mouth corners	extra low, low, average, raised, extra raised
mouth height	extra thick, thick, average, thin, extra thin
mouth open	yes, no
mouth width	extra wide, wide, average, small, extra small
mustache	yes, no
narrow eyes	yes, no
nose shape	extra straight, straight, average, triangle, extra triangle
nose width	extra wide, wide, average, narrow, extra narrow
oval face	yes, no
pale skin	yes, no
pointy nose	yes, no
race	asian-middle-eastern, asian, african-american, hispanic, white, middle eastern, other
receding hairline	yes, no
rosy cheeks	rosy cheeks
smile	yes, no
sideburns	yes, no
straight hair	yes, no
teeth visible	yes, no
wavy hair	yes, no
wearing earrings	yes, no
wearing hat	yes, no
wearing lipstick	yes, no
wearing necklace	yes, no
wearing necktie	yes, no
young	yes, no

Appendix 2: Possible classifiers with return values Microsoft Azure

In the table below, all possible classifiers in the category “face attributes”, as retrieved from https://docs.microsoft.com/en-us/rest/api/cognitiveservices/face/face/detectwithstream#faceattributes on 2 February 2019, are listed with their corresponding return values.

Classifier	**Return value**
accessories	glasses, headWear, mask
age	0 - 100
blur	low, medium, high
emotion	anger, contempt, disgust, fear, happiness, neutral, sadness, surprise
exposure	0 - 1
facial hair	moustache, sideburns, beard
gender	male, female
glasses	noGlasses, readingGlasses, sunGlasses
hair	bald, hairColor, invisible
headPose	pitch, roll, yaw
makeup	boolean eye make-up, boolean lip make-up
noise	0 - 1
occlusion	eyeOccluded, forheadOccluded, mouthOccluded
smile	0 - 1



Appendix 3: All 64 hairstyles

Source: Yin, Weidong, et al. "Learning to generate and edit hairstyles." Proceedings of the 2017 ACM on Multimedia Conference. ACM, 2017.

Appendix 4:Codes

Python code for extracting data from Microsoft Azure Face API

```
# coding: utf-8

# In[ ]:


import cognitive_face as CF
import time
import csv
import json

hairstyle = ['Aaron_Kwok', 'Afro', 'Bald', 'Beehive',
'Bob', 'Bouffant', 'Bowl_Cut', 'Bun', 'Caesar',
'Chonmage', 'Comb_Over', 'Cornrows', 'Crew_Cut', 'Crop',
'Croydon_Facelift', 'Curly', 'Curly_Hair',
'Curtained_Hair', 'Cute_Ponytails', 'Devilock',
'Dreadlocks', 'Ducktail', 'Emo_hair', 'Fauxhawk',
'Flattop', 'French_Braid', 'French_Twist', 'Hi-top_Fade',
'Hime_Cut', 'Horseshoe_Flattop', 'Induction_Cut',
'Jimmy_Lin_Hairstyle', 'Layered_Hair',
'Liberty_Spikes_Hair', 'Long_Hair', 'Medium-Length_Hair',
'Men_Pompadour', 'Men_With_Square_Angles', 'Mohawk',
'Mop-Top_Hair', 'Mullet', 'Odango_Hair', 'Pageboy',
'Perm', 'Pixie_Cut', 'Ponytail', 'Quiff', 'Rattail',
'Razor_Cut', 'Ringlet', 'Shag', 'Shoulder-Length_Hair',
'Side_Part', 'Slicked-back', 'Spiky_Hair',
'Tapered_Sides', 'The_Rachel', 'Tonsure_Hair', 'Updo',
'Waist-Length_Hair', 'Wave_Hair']
num = [105, 386, 66, 383, 603, 111, 203, 404, 103, 40,
47, 328, 205, 109, 50, 62, 926, 351, 490, 21, 652, 19,
67, 84, 310, 162, 78, 243, 90, 22, 81, 51, 249, 118, 11,
225, 99, 43, 104, 97, 184, 63, 259, 284, 499, 308, 22,
30, 381, 117, 411, 461, 92, 178, 368, 52, 33, 32, 108,
82, 254]

KEY = '5f1a14d290b54640804ee44a94548c66'  # Replace with
a valid Subscription Key here.
CF.Key.set(KEY)

BASE_URL =
'https://northeurope.api.cognitive.microsoft.com/face/v1.
0'  # Replace with your regional Base URL
CF.BaseUrl.set(BASE_URL)

f = csv.writer(open("test.csv", "w"))
f.writerow(["Hairstyle", "Pic", "Gender", "Age",
"Moustache", "Beard", "Sideburns", "Glasses"])

for i in range(61):
    img_url = []
    for j in range(num[i]):
img_url.append('https://raw.githubusercontent.com/sarkrui
/Hairstyle60k/master/Dataset/' + hairstyle[i] + '/IMG_'+
str(j+1) + '.jpg')
        result = CF.face.detect(img_url[j], face_id=True,
landmarks=False,
attributes='age,gender,facialHair,glasses')
        print (result)
        result = json.dumps(result)
        result = json.loads(result)
        f = csv.writer(open("test.csv", "a"))
        if result == []:
            f.writerow([hairstyle[i], 'IMG_'+str(j+1), 0,
0, 0, 0, 0, 0])
        else:
            for result in result:
                f.writerow([hairstyle[i],
'IMG_'+str(j+1), result["faceAttributes"]["gender"],
result["faceAttributes"]["age"],
result["faceAttributes"]["facialHair"]["moustache"],
result["faceAttributes"]["facialHair"]["beard"],
result["faceAttributes"]["facialHair"]["sideburns"],
result["faceAttributes"]["glasses"]])
        time.sleep(3)
```

Shell script for extracting data from Betaface

```
#!/bin/sh
cd "$(dirname "$0")"

INDEX="1"

declare -a STYLE=("Aaron_Kwok" "Afro" "Bald" "Beehive"
"Bob" "Bouffant" "Bowl_Cut" "Bun" "Caesar" "Chonmage"
"Comb_Over" "Cornrows" "Crew_Cut" "Crop"
"Croydon_Facelift" "Curly" "Curly_Hair" "Curtained_Hair"
"Cute_Ponytails" "Devilock" "Dreadlocks" "Ducktail"
"Emo_hair" "Fauxhawk" "Flattop" "French_Braid"
"French_Twist" "Hi-top_Fade" "Hime_Cut"
"Horseshoe_Flattop" "Induction_Cut" "Jimmy_Lin_Hairstyle"
"Layered_Hair" "Liberty_Spikes_Hair" "Long_Hair" "Medium-
Length_Hair" "Men_Pompadour" "Men_With_Square_Angles"
"Mohawk" "Mop-Top_Hair" "Mullet" "Odango_Hair" "Pageboy"
"Perm" "Pixie_Cut" "Ponytail" "Quiff" "Rattail"
"Razor_Cut" "Ringlet" "Shag" "Shoulder-Length_Hair"
"Side_Part" "Slicked-back" "Spiky_Hair" "Tapered_Sides"
"The_Rachel" "Tonsure_Hair" "Updo" "Waist-Length_Hair"
"Wave_Hair")
declare -a COUNT=("105" "386" "66" "383" "603" "111"
"203" "404" "103" "40" "47" "328" "205" "109" "50" "62"
"926" "351" "490" "21" "652" "19" "67" "84" "310" "162"
"78" "243" "90" "22" "81" "51" "249" "118" "11" "225"
"99" "43" "104" "97" "184" "63" "259" "284" "499" "308"
"22" "30" "381" "117" "411" "461" "92" "178" "368" "52"
"33" "32" "108" "82" "254")
```

```
for (( i = 0; i < 61; i++ )); do


        #Insert a opening bracket JSON Array
        printf "[\n" >> ${STYLE[$i]}.json


        NUMBER="1"
        for (( j = 0; j < ${COUNT[$i]} ; j++ )); do

                #printf
"https://raw.githubusercontent.com/sarkrui/Hairstyle60k/m
aster/Dataset/${STYLE[$i]}/IMG_$INDEX.jpg"
                curl -sS
https://www.betafaceapi.com/api/v2/media -H "accept:
application/json" -H "Content-Type: application/json" -d
"{ \"api_key\": \"d45fd466-51e2-4701-8da8-04351c872236\",
\"file_uri\":
\"https://raw.githubusercontent.com/sarkrui/Hairstyle60k/
master/Dataset/${STYLE[$i]}/IMG_$NUMBER.jpg\",
\"detection_flags\": \"classifiers,extended\"}" >>
${STYLE[$i]}.json
                printf ${STYLE[$i]}_IMG_$NUMBER-
                printf INDEX_$INDEX"\n"
                printf ",\n" >> ${STYLE[$i]}.json
                let "INDEX+=1"
                let "NUMBER+=1"

        if [[ $INDEX -eq 499 ]]; then
                afplay /System/Library/Sounds/Funk.aiff
                read -p "Please reset your ip...Press
[ENTER] to continute."
                INDEX="1"
        fi

        done
        #Removing the comma one the last line
        sed -i "_bak" '$ s/,$//g' ${STYLE[$i]}.json

        #Insert a JSON Array closing bracket
        printf "]\n" >> ${STYLE[$i]}.json

done
exit
```

Java code for data validation

```
import org.neuroph.core.*;
import org.neuroph.core.data.*;
import org.neuroph.core.data.norm.*;
import org.neuroph.core.data.sample.*;
import org.neuroph.core.events.*;
import org.neuroph.core.exceptions.*;
import org.neuroph.core.input.*;
import org.neuroph.core.learning.error.*;
import org.neuroph.core.learning.*;
import org.neuroph.core.learning.stop.*;
import org.neuroph.core.transfer.*;
import org.neuroph.nnet.*;
import org.neuroph.nnet.comp.*;
import org.neuroph.nnet.comp.layer.*;
import org.neuroph.nnet.comp.neuron.*;
import org.neuroph.nnet.learning.*;
import org.neuroph.util.benchmark.*;
import org.neuroph.util.*;
import org.neuroph.util.io.*;
import org.neuroph.util.plugins.*;
import org.neuroph.util.random.*;

import java.util.*;

String str = new String();

Table table;
Table validation;
int w = 0;

String output[] =
{"Aaron_Kwok","Afro","Bald","Beehive","Bob","Bouffant","B
owl_Cut","Bun","Caesar","Chonmage","Comb_Over","Cornrows"
,"Crew_Cut","Crop","Croydon_Facelift","Curly","Curly_Hair
","Curtained_Hair","Cute_Ponytails","Dreadlocks","Emo_hai
r","Fauxhawk","Flattop","French_Braid","Hi-
top_Fade","Hime_Cut","Induction_Cut","Jimmy_Lin_Hairstyle
","Layered_Hair","Liberty_Spikes_Hair","Medium-
Length_Hair","Men_Pompadour","Men_With_Square_Angles","Mo
hawk","Mop-
Top_Hair","Mullet","Odango_Hair","Pageboy","Perm","Pixie_
Cut","Ponytail","Razor_Cut","Ringlet","Shag","Shoulder-
Length_Hair","Side_Part","Slicked-
back","Spiky_Hair","Tapered_Sides","The_Rachel","Updo","W
aist-Length_Hair","Wave_Hair"};
String[] topTen = new String[10];
float[] topData = new float[10];

void setup() {
  size(1400,720);
  table = loadTable("Validation.csv", "header");
  validation = new Table();

  validation.addColumn("Lable");
  validation.addColumn("First Choice");
  validation.addColumn("Second Choice");
  validation.addColumn("Third Choice");
  validation.addColumn("Fourth Choice");
  validation.addColumn("Fifth Choice");
  validation.addColumn("Sixth Choice");
  validation.addColumn("Seventh Choice");
  validation.addColumn("Eighth Choice");
  validation.addColumn("Nineth Choice");
  validation.addColumn("Tenth Choice");

  // load saved neural network
  NeuralNetwork neuralNetwork1 =
NeuralNetwork.createFromFile(sketchPath("myMlPerceptron.n
net"));

  // set network input
  for (TableRow row : table.rows()) {
    w = w + 1;
    float[] data_row = new float[26];
    for(int i = 0; i < 26; i++){
      data_row[i] = row.getFloat(str(i+1));
    }

neuralNetwork1.setInput(data_row[0],data_row[1],data_row[
2],data_row[3],data_row[4],data_row[5],data_row[6],data_r
ow[7],data_row[8],data_row[9],data_row[10],data_row[11],d
ata_row[12],data_row[13],data_row[14],data_row[15],data_r
ow[16],data_row[17],data_row[18],data_row[19],data_row[20
],data_row[21],data_row[22],data_row[23],data_row[24],dat
a_row[25]);

    // calculate network
    neuralNetwork1.calculate();

    // get network output
    double[] networkOutput = neuralNetwork1.getOutput();

    str = Arrays.toString(networkOutput);

    // get the top ten hairstyle:
    for(int i = 0; i < 53; i ++ ){
      int n = 0;
      for(int j = 0; j < 53; j ++){
        if(networkOutput[i] < networkOutput[j])
        n ++;
      }
      if(n == 0){
        topTen[0] = output[i];
        topData[0] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 1){
        topTen[1] = output[i];
        topData[1] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 2){
        topTen[2] = output[i];
        topData[2] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 3){
        topTen[3] = output[i];
        topData[3] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 4){
        topTen[4] = output[i];
        topData[4] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 5){
        topTen[5] = output[i];
        topData[5] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 6){
        topTen[6] = output[i];
        topData[6] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 7){
        topTen[7] = output[i];
        topData[7] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 8){
        topTen[8] = output[i];
        topData[8] = (float)networkOutput[i];
        n = 0;
      }
      else if(n == 9){
        topTen[9] = output[i];
        topData[9] = (float)networkOutput[i];
        n = 0;
      }
      else n = 0;
    }

    println("running time:" + w);

    TableRow newRow = validation.addRow();
    newRow.setString("Lable", output[(w-1)/3]);
    newRow.setString("First Choice", topTen[0]);
    newRow.setString("Second Choice", topTen[1]);
    newRow.setString("Third Choice", topTen[2]);
    newRow.setString("Fourth Choice", topTen[3]);
    newRow.setString("Fifth Choice", topTen[4]);
    newRow.setString("Sixth Choice", topTen[5]);
    newRow.setString("Seventh Choice", topTen[6]);
    newRow.setString("Eighth Choice", topTen[7]);
    newRow.setString("Nineth Choice", topTen[8]);
    newRow.setString("Tenth Choice", topTen[9]);

    saveTable(validation, "data/new.csv");
  }
}

void draw(){
  textSize(30);
  fill(0);
  textAlign(CENTER);
  text("HAIR RECOMMENDATION", 1000, 50);

  textSize(40);
  fill(0);
  textAlign(CENTER);
  text(topTen[0]+": "+topData[0], 1000, 160);
  text(topTen[1]+": "+topData[1], 1000, 290);
  text(topTen[2]+": "+topData[2], 1000, 420);

  textSize(10);
  textAlign(LEFT);
  text(str, 580, 520, 800, 720);
}
```

Java code as shown at our final presentation

```
import org.neuroph.core.*;
import org.neuroph.core.data.*;
import org.neuroph.core.events.*;
import org.neuroph.core.exceptions.*;
import org.neuroph.core.input.*;
import org.neuroph.core.learning.error.*;
import org.neuroph.core.learning.*;
import org.neuroph.core.learning.stop.*;
import org.neuroph.core.transfer.*;
import org.neuroph.eval.classification.*;
import org.neuroph.eval.*;
import org.neuroph.nnet.*;
import org.neuroph.nnet.comp.*;
import org.neuroph.nnet.comp.layer.*;
import org.neuroph.nnet.comp.neuron.*;
import org.neuroph.nnet.learning.*;
import org.neuroph.nnet.learning.kmeans.*;
import org.neuroph.nnet.learning.knn.*;
import org.neuroph.util.benchmark.*;
import org.neuroph.util.*;
import org.neuroph.util.data.norm.*;
import org.neuroph.util.data.sample.*;
import org.neuroph.util.io.*;
import org.neuroph.util.plugins.*;
import org.neuroph.util.random.*;
import http.requests.*;
import java.util.*;

//curl -v -X POST
"https://westcentralus.api.cognitive.microsoft.com/face/v
1.0/detect?returnFaceId=true&returnFaceLandmarks=false&re
turnFaceAttributes={string}"
//-H "Content-Type: application/json"
//-H "Ocp-Apim-Subscription-Key: {subscription key}"
//--data-ascii "{body}"

import http.requests.*;
PrintWriter output;
JSONArray msResponse;
JSONArray betafaceResponse;
JSONObject faceAttributes;


float[] remapped = new float[20];
int remapped_gender = 0;
float remapped_glasses = 0;
float remapped_age;
float sideburns = 0;
float beard = 0;
float moustache = 0;

void getMS() {

  msResponse = new JSONArray();
  faceAttributes = new JSONObject();

  //Creates a json file in the sketch directory to store
JSON response from Microsoft FaceAPI
  output = createWriter("msResponse.json");

  //HTTP Request body
  PostRequest post = new
PostRequest("https://northeurope.api.cognitive.microsoft.
com/face/v1.0/detect?returnFaceId=true&returnFaceLandmark
s=false&returnFaceAttributes=gender,age,facialHair,glasse
s");
  post.addHeader("Host",
"northeurope.api.cognitive.microsoft.com");
  post.addHeader("Content-Type", "application/json");
  post.addHeader("Ocp-Apim-Subscription-Key",
"5f1a14d290b54640804ee44a94548c66");

post.addJson("{\"url\":\"https://raw.githubusercontent.co
m/sarkrui/Hairstyle60k/master/sample_3.jpg\"}");
  post.send();

  //Parse HTTP Response as a JSON Array
  JSONArray msResponse =
parseJSONArray(post.getContent());

  //Saves JSON Array to local for backup
  saveJSONArray(msResponse, "data/msResponse.json");

  //Reads local JSON file
  //msResponse = loadJSONArray("data/msResponse.json");
//"Gender","Age","Moustache","Beard","Sideburns","Glasses
"

  //Expands JSONArray msResponse[0]
  JSONObject msObject = msResponse.getJSONObject(0);

  //Expands JSONObject faceAttributes
  JSONObject faceAttributes =
msObject.getJSONObject("faceAttributes");

  //Expands JSONObject facialHair
  JSONObject facialHair =
faceAttributes.getJSONObject("facialHair");

  //Expands JSONObject faceAttributes
  String glasses = faceAttributes.getString("glasses");
  String gender = faceAttributes.getString("gender");
  int age = faceAttributes.getInt("age");

  //Retrieves a String value from
  sideburns = facialHair.getFloat("sideburns");
  beard = facialHair.getFloat("beard");
  moustache = facialHair.getFloat("moustache");

  //gender:
  if (gender == "female") {
    remapped_gender = 0;
  } else if (gender == "male") {
    remapped_gender = 1;
  }

  //glasses:
  if (glasses == "noGlasses") {
    remapped_glasses = 0;
  } else if (glasses == "ReadingGlasses") {
    remapped_glasses = 0.5;
  } else {
    remapped_glasses = 1;
  }

  //age:
  remapped_age = age/100.0;

  //JSONObject faceAttributes =
MS_Object.getJSONObject(1);
  //String Age = faceAttributes.getString("glasses");
  //String Moustache =
faceAttributes.getString("moustache");
  println(remapped_gender, "\n", remapped_age, "\n",
moustache, "\n", beard, "\n", sideburns, "\n",
remapped_glasses);
}
```

```
//curl -sS https://www.betafaceapi.com/api/v2/media -H
"accept: application/json" -H "Content-Type:
application/json" -d "{ \"api_key\": \"d45fd466-51e2-
4701-8da8-04351c872236\", \"file_uri\":
\"https://raw.githubusercontent.com/sarkrui/Hairstyle60k/
master/Dataset/${STYLE[$i]}/IMG_$INDEX.jpg\",
\"detection_flags\": \"classifiers\"}"
//https://raw.githubusercontent.com/sarkrui/Hairstyle60k/
master/Dataset/Curly/IMG_39.jpg
//import http.requests.*;

void getBetaface() {

  betafaceResponse = new JSONArray();

  //Declares the reading order from tags
  int[] readingOrder = {27, 0, 2, 4, 6, 7, 8, 9, 14, 16,
22, 23, 26, 28, 35, 36, 44, 54, 66, 69};

  PostRequest post = new PostRequest("https:" +
"//www.betafaceapi.com/api/v2/media");
  post.addHeader("accept", "application/json");
  post.addHeader("Content-Type", "application/json");
  post.addJson("{\"api_key\": \"d45fd466-51e2-4701-8da8-
04351c872236\",\"file_uri\":
\"https://raw.githubusercontent.com/sarkrui/Hairstyle60k/
master/sample_3.jpg\",\"detection_flags\":
\"classifiers,extended\"}");
  post.send();

  //System.out.println(post.getContent() + "\n");

  //Parse HTTP Response as a JSON Array
  JSONObject betafaceResponse =
parseJSONObject(post.getContent());

  //Saves JSON Array to local
  saveJSONObject(betafaceResponse,
"data/betafaceResponse.json");

  //Expands JSONObject media
  JSONObject media =
betafaceResponse.getJSONObject("media");

  //Store JSONArray faces
  JSONArray faces = media.getJSONArray("faces");

  //Expands JSONArray faces
  JSONObject face = faces.getJSONObject(0);

  //Expands JSONObject face
  JSONArray tags = face.getJSONArray("tags");

  //float[] remapped = new float[20];
  for (int i = 0; i < 20; i++) {

    //Expands JSONArray tags
    JSONObject feature =
tags.getJSONObject(readingOrder[i]);

    //Expands JSONObject feature
    float confidence = feature.getFloat("confidence");
    String name = feature.getString("name");
    String value = feature.getString("value");

    if (value == "yes") {
      remapped[i] = 0.5 + confidence/2.0;
    } else if (value == "no") {
      remapped[i] = 0.5 - confidence/2.0;
    } else {
      remapped[i] = confidence;
    }

    if (value == "extra small") {
      remapped[i] = 0;
    } else if (value == "small") {
      remapped[i] = 0.25;
    } else if (value == "average") {
      remapped[i] = 0.5;
    } else if (value == "big") {
      remapped[i] = 0.75;
    } else if (value == "extra big") {
      remapped[i] = 1;
    } else {
      remapped[i] = confidence;
    }

    if (value == "extra narrow") {
      remapped[i] = 0;
    } else if (value == "narrow") {
      remapped[i] = 0.25;
    } else if (value == "average") {
      remapped[i] = 0.5;
    } else if (value == "wide") {
      remapped[i] = 0.75;
    } else if (value == "extra wide") {
      remapped[i] = 1;
    } else {
      remapped[i] = confidence;
    }

    if (value == "extra close") {
      remapped[i] = 0;
    } else if (value == "close") {
      remapped[i] = 0.25;
    } else if (value == "average") {
      remapped[i] = 0.5;
    } else if (value == "far") {
      remapped[i] = 0.75;
    } else if (value == "extra far") {
      remapped[i] = 1;
    } else {
      remapped[i] = confidence;
    }

    println(name, remapped[i]);
  }
}
// image on GitHub
String url =
"https://raw.githubusercontent.com/sarkrui/Hairstyle60k/m
aster/sample_3.jpg";
PImage webImg;
String str = new String();

PImage one;
PImage two;
PImage three;

// the three output labels with highest output will be
shown
String output_category[] = {"Aaron_Kwok", "Afro", "Bald",
"Beehive", "Bob", "Bouffant", "Bowl_Cut", "Bun",
"Caesar", "Chonmage", "Comb_Over", "Cornrows",
"Crew_Cut", "Crop", "Croydon_Facelift", "Curly",
"Curly_Hair", "Curtained_Hair", "Cute_Ponytails",
"Dreadlocks", "Emo_hair", "Fauxhawk", "Flattop",
"French_Braid", "Hi-top_Fade", "Hime_Cut",
"Induction_Cut", "Jimmy_Lin_Hairstyle", "Layered_Hair",
"Liberty_Spikes_Hair", "Medium-Length_Hair",
"Men_Pompadour", "Men_With_Square_Angles", "Mohawk",
"Mop-Top_Hair", "Mullet", "Odango_Hair", "Pageboy",
"Perm", "Pixie_Cut", "Ponytail", "Razor_Cut", "Ringlet",
"Shag", "Shoulder-Length_Hair", "Side_Part", "Slicked-
back", "Spiky_Hair", "Tapered_Sides", "The_Rachel",
"Updo", "Waist-Length_Hair", "Wave_Hair"};
String[] topThree = new String[3];
float[] topData = new float[3];

void setup() {
  size(1400, 720);
  webImg = loadImage(url, "jpg");

  getMS();
  getBetaface();

  // load saved neural network
  NeuralNetwork neuralNetwork1 =
NeuralNetwork.createFromFile(sketchPath("SarkNN_Win.nnet"
));

  // set network input
  neuralNetwork1.setInput(remapped_gender, remapped_age,
moustache, beard, sideburns, remapped_glasses,
remapped[0], remapped[1], remapped[2], remapped[3],
remapped[4], remapped[5], remapped[6], remapped[7],
remapped[8], remapped[9], remapped[10], remapped[11],
remapped[12], remapped[13], remapped[14], remapped[15],
remapped[16], remapped[17], remapped[18], remapped[19]);
  // calculate network
  neuralNetwork1.calculate();

  // get network output
  double[] networkOutput = neuralNetwork1.getOutput();
  System.out.println(" Output: " +
Arrays.toString(networkOutput) );
  str = Arrays.toString(networkOutput);

  for (int i = 0; i < 53; i ++ ) {
    int n = 0;
    for (int j = 0; j < 53; j ++) {
      if (networkOutput[i] < networkOutput[j])
        n ++;
    }
    if (n == 0) {
      topThree[0] = output_category[i];
      topData[0] = (float)networkOutput[i];
      n = 0;
    } else if (n == 1) {
      topThree[1] = output_category[i];
      topData[1] = (float)networkOutput[i];
      n = 0;
    } else if (n == 2) {
      topThree[2] = output_category[i];
      topData[2] = (float)networkOutput[i];
      n = 0;
    } else n = 0;
  }

  for (int i = 0; i < 3; i ++) {
    println(topThree[i]);
  }
}

void draw() {
  //image(img, 0, 0, 540, 720);
  image(webImg, 0, 0, 540, 720);

  one = loadImage(topThree[0]+".jpg");
  two = loadImage(topThree[1]+".jpg");
  three = loadImage(topThree[2]+".jpg");

  image(one, float(600), float(120), float(100),
float(100));
  image(two, float(600), float(250), float(100),
float(100));
  image(three, float(600), float(380), float(100),
float(100));

  textSize(50);
  fill(0);
  textAlign(CENTER);
  text("HAIR RECOMMENDATION", 1000, 50);

  textSize(40);
  fill(0);
  textAlign(CENTER);
  text(topThree[0]+": "+topData[0], 1000, 160);
  text(topThree[1]+": "+topData[1], 1000, 290);
  text(topThree[2]+": "+topData[2], 1000, 420);

  textSize(10);
  textAlign(LEFT);
  text(str, 580, 520, 800, 720);
}


//Tab postrequest
package http.requests;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map.Entry;

import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import
org.apache.http.client.entity.UrlEncodedFormEntity;
import
org.apache.http.client.methods.HttpEntityEnclosingRequest
Base;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.ByteArrayEntity;
import org.apache.http.entity.StringEntity;
import org.apache.http.entity.mime.MultipartEntity;
import org.apache.http.entity.mime.content.FileBody;
import org.apache.http.entity.mime.content.StringBody;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicHeader;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;

public class PostRequest
```

```
{
  String url;
  ArrayList<BasicNameValuePair> nameValuePairs;
  HashMap<String,File> nameFilePairs;
  List<Header> headers;

  String method;
  String content;
  String encoding;
  HttpResponse response;
  String json;
  byte[] binary;

  public PostRequest(String url)
  {
    this(url, "ISO-8859-1");
  }

  public PostRequest(String url, String encoding)
  {
    this.url = url;
    this.encoding = encoding;
    nameValuePairs = new ArrayList<BasicNameValuePair>();
    nameFilePairs = new HashMap<String,File>();
    headers = new ArrayList<Header>();
  }

  public void addData(String key, String value)
  {
    BasicNameValuePair nvp = new
BasicNameValuePair(key,value);
    nameValuePairs.add(nvp);
  }

  public void addData(byte[] binary) {
    addData(null, binary);
  }
  public void addData(String contentType, byte[] binary)
{
    if (contentType != null) {
      addHeader(HttpHeaders.CONTENT_TYPE, contentType);
    }
    this.binary = binary;
  }

  public void addDataFromFile(String fullPathname) {
    addDataFromFile(null, fullPathname);
  }
  public void addDataFromFile(String contentType, String
fullPathname) {
    if (contentType != null) {
      addHeader(HttpHeaders.CONTENT_TYPE, contentType);
    }
    Path path = Paths.get(fullPathname);
    System.out.println("Path: " + path.toAbsolutePath());
    try {
      this.binary = Files.readAllBytes(path);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public void addJson(String json) {
    addHeader(HttpHeaders.CONTENT_TYPE,
"application/json");
    this.json = json;
  }

  public void addFile(String name, File f) {
    nameFilePairs.put(name,f);
  }

  public void addFile(String name, String path) {
    File f = new File(path);
    nameFilePairs.put(name,f);
  }

  public void addHeader(String name, String value) {
    headers.add(new BasicHeader(name, value));
  }

  // only PUT will change anything, anything else
defaults to POST
  public void method(String put) {
    if (put != null && put.equalsIgnoreCase("PUT")) {
      this.method = put;
    }
  }

  public void send()
  {
    try {
      DefaultHttpClient httpClient = new
DefaultHttpClient();
      HttpEntityEnclosingRequestBase httpPost;

      // you can specify this is a PUT request.
everything else is a POST.
      if (method != null &&
method.equalsIgnoreCase("PUT")) {
        httpPost = new HttpPut(url);
      } else {
        httpPost = new HttpPost(url);
      }

      if (!nameValuePairs.isEmpty()) {
        httpPost.setEntity(new
UrlEncodedFormEntity(nameValuePairs, encoding));
      }
      // add binary
      else if (binary != null) {
        httpPost.setEntity(new ByteArrayEntity(binary));
      }
      // add json
      else if (json != null) {
        httpPost.setEntity(new StringEntity(json));
      }
      // file handling
      else if (!nameFilePairs.isEmpty()) {
        MultipartEntity mentity = new MultipartEntity();
        Iterator<Entry<String, File>> it =
nameFilePairs.entrySet().iterator();
        while (it.hasNext()) {
          Entry<String, File> pair = it.next();
          String name = pair.getKey();
          File f = pair.getValue();
          mentity.addPart(name, new FileBody(f));
        }
        for (NameValuePair nvp : nameValuePairs) {
          mentity.addPart(nvp.getName(), new
StringBody(nvp.getValue()));
        }
        httpPost.setEntity(mentity);
      }

      // add the headers to the request
      if (!headers.isEmpty()) {
        for (Header header : headers) {
          httpPost.addHeader(header);
        }
      }

      response = httpClient.execute(httpPost);
      HttpEntity entity = response.getEntity();
      this.content =
EntityUtils.toString(response.getEntity());

      if( entity != null ) EntityUtils.consume(entity);

      httpClient.getConnectionManager().shutdown();

      // Clear it out for the next time
      nameValuePairs.clear();
      nameFilePairs.clear();
      headers.clear();
      json = null;
      binary = null;
      method = null;

    } catch( Exception e ) {
      e.printStackTrace();
    }
  }

  /* Getters
_________________________________________________________
____ */

  public String getContent()
  {
    return this.content;
  }

  public String getHeader(String name)
  {
    Header header = response.getFirstHeader(name);
    if(header == null)
    {
      return "";
    }
    else
    {
      return header.getValue();
    }
  }
}
```