Computer Vision Project Final Report

Team Okurki

Aleksandr Mikhailov
al.mikhailov@innopolis.university

Makar Egorov
m.egorov@innopolis.university

Ivan Chabanov
i.chabanov@innopolis.university

I. Links

- Backend repository
  https://github.com/okurki/beauty-classifier
- Frontend repository
  https://github.com/okurki/app
- Project Backlog
  https://github.com/users/ChabanovX/projects/5/views/1
- Backend
  http://10.90.137.195:8000
- Mlflow server
  http://10.90.137.195:5000

II. Project Topoic

A mobile application that uses computer vision to analyze facial features for entertainment purposes. The app provides two main functionalities:

- Facial Attractiveness Score: Estimates an attractiveness rating based on facial features
- Celebrity Look-alike Finder: Identifies which famous celebrity the user resembles most
- Adaptive celebrity matching: Look-alike Finder model employs reinforcement learning to adapt to user feedback

A. Importance and relevance

From a technical point of view, the project serves as a showcase of modern machine learning and computer vision techniques, mastery of the entire ML pipeline, starting from data collection to deployment to a REST API.

From user perspective, the system serves as a ”digital mirror”, utilizing self-perception and social comparison. It serves as a personalized engagement tool, offering entertainment and self-reflection.

The core ML models could be fine-tuned and repurposed for facial recognition or authentication applications, or extracting other features from faces apart from the ”attractiveness score”, or used ”as is” in dating apps, for example.

III. Related Work

Facial analysis has been extensively studied in computer vision research. For celebrity look-alike recognition, systems typically use deep metric learning approaches [4] with pre-trained models like FaceNet or VGG-Face to extract facial embeddings, then compute similarity scores against celebrity databases [1].

For facial attractiveness prediction, recent approaches have used hybrid CNN-Transformer architectures to capture multi-scale facial features. However, most public datasets like SCUT-FBP5500 have limitations in racial diversity, which can lead to biased models [2].

Commercial applications like Gradient and Celebs have demonstrated the popularity of celebrity look-alike features, while apps like FaceApp show user interest in AI-based facial analysis for entertainment.

A. Celebrity Look-alike Apps

Gradient: A celebrity face match app that uses advanced AI-powered face recognition. It has over 10 million users and ratings of 4.1 on iOS and 3.5 on Android.

Celebs: Focuses on fast and accurate celebrity matching with a database of over 2,000 celebrities. It includes integrated AI tools for enhanced matching accuracy and social media sharing capabilities. The app has been downloaded over 8 million times and maintains a 4.3 rating on iOS.

Look Alike – Celebrity: Known for its minimalist design and straightforward functionality, it provides fast results with an extensive celebrity database. It has attracted over 5 million users with ratings of 4.1 on iOS and 3.3 on Androd.

Celeb Twin: This app provides a detailed percentage resemblance score and features a database of over 3000 celebrities. It includes social features that allow users to compare matches with friends and share results on platforms like Instagram and WhatsAp.

B. Facial Analysis and Beauty Scoring Applications

While pure attractiveness scoring apps are less common, several applications incorporate related functionalit:

FaceApp – AI Face Editor: Primarily an entertainment-focused photo editor, it includes beauty filters that enhance attractiveness by modifying smiles, hairstyles, makeup, and other features. Its popularity demonstrates user interest in AI-modified appearance enhancement.

Face Secret – Face Reading: Uses facial features to provide entertainment-based readings according to horoscope principles. While not scientifically validated, its popularity suggests user interest in deriving personal insights from facial analyss.

C. Technical Approaches and SOTA Solutios

Deep Learning for Celebrity Look-Alike:

Systems like those in [1] use CNNs to represent faces as vector representations. Distance metrics (e.g., cosine similarity) are used to find the closest matches in a celebrity database.

Pre-trained models like VGG-Face and FaceNet are commonly used for feature extraction.

Facial Beauty Prediction (FBP): Recent advancements use hybrid CNN-Transformer models (e.g., Scale-Interaction Transformer) to capture multi-scale facial features and model interactions between them. These approaches achieve high Pearson correlation coefficients (e.g., 0.9187 on benchmark datasets) by integrating local and global facial feaurs [1].

Challenges and Limitations: Current systems face issues with bias and representation [2]. For example, celebrity databases are often skewed toward Western celebrities, leading to poor matches for non-Western users. Additionally, attractveness scoring systems can perpetuate Eurocentric beauty standards if not carefully designe.

IV. Methodology

A. System Architecture

We built a three-tier architecture:

- **Mobile Frontend:** iOS app for image capture and results display on Flutter [12]
- **Backend API:** Python/FastAPI service handling model inference and user management
- **ML Infrastructure:** Mlflow for experiment tracking, DVC for data versioning
- **Hosting:** Mlflow server and the API are hosted on a university-provided virtual machine with docker compose deployment [15].

B. Datasets

We used two primary datasets:

- **SCUT-FBP5500:** For attractiveness prediction, containing 5500 facial images with attractiveness scores (1-5)
- **Open Famous People Faces:** For celebrity matching, containing 100,000+ images across 10,000+ celebrities

C. SCUT-FBP5500 dataset

The source is available on GitHub [3].

As been stated in the previous section, the dataset turned out to be balanced and well-split.

Dataset had no missing values, which simplified the processing pipeline.

We decided to normalize the scores from 1.0 $\rightarrow$ 5.0 to 0.0 $\rightarrow$ 1.0 and separately save them.

IV-D shows distribution of score between train and test sets.

D. Open Famous People Faces dataset

The source is available on Kaggle.

It is organized with subfolders named after a celebrity containg their images.

Since it is an image-based dataset, we analyzed image count distribution per celebrity and image sizes. IV-D shows distribution histogram.

The images are of high quality and are properly centered and cropped. This made us think if it'd be good to add a preprocessing step with centering and cropping for the SCUT-FBP5500 dataset pipeline.

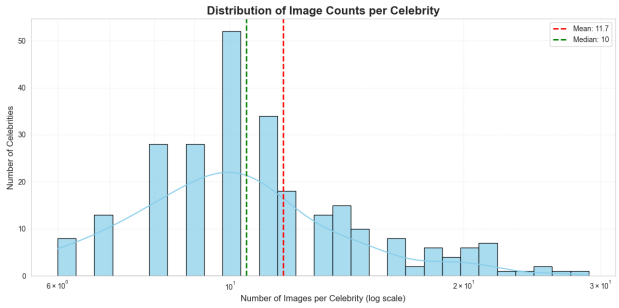

Fig. 1. OFPF Image counts histogram

```
--- Comparing Train and Test Distributions ---
Train set size: 3300
Test set size: 2200

Train Set Scores - Describe:
count    3300.000000
mean        0.497198
std         0.173632
min         0.008333
25%         0.375000
50%         0.458333
75%         0.633333
max         0.925000
Name: score, dtype: float64

Test Set Scores - Describe:
count    2200.000000
mean        0.498509
std         0.169630
min         0.004167
25%         0.375000
50%         0.462500
75%         0.629167
max         0.937500
Name: score, dtype: float64
```

Fig. 2. SCUT Train and test metrics

Although, as pointed out to us, it does contain images of all races, while SCUT-FBP5500 dataset only contains images of caucasian and asian people, which can lead to biases or OOD in our models.

We justify our choice of this dataset combination by the fact that they are well-tested and easy to use. The problematic dataset is SCUT-FBP5500 as there are no well-maintained datasets for face attractiveness, so we have to resort to this one.

Concerning the issues with model biases or OOD, we decided that it's ok to have these imperfections as this is not a production system and only a proof of concept. If we decide to expand our work on this project, it is entirely possible to manually collect an attractiveness dataset with all race examples.

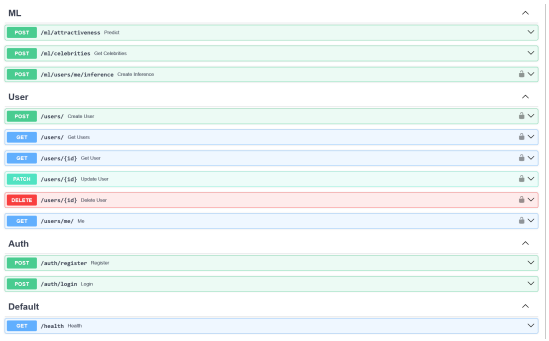

Fig. 3. Swagger API docs

E. Models

AttractivenessClassifier: Fine-tuned ResNet-50 architecture for regression based on pytorch [11] implementation, trained to predict normalized attractiveness scores (0-1). We used transfer learning and data augmentation to improve generalization based on [14]'s method.

LookALikeFinder: Based on InceptionResnetV1 with metric learning, using triplet loss to learn facial embeddings. The model computes cosine similarity between user faces and celebrity embeddings.

V. Experiments and Metrics

Experiments and metrics were collected using Mlflow [8]. Our most recent runs showed

A. Technical Challenges

Dataset Limitations: The SCUT-FBP5500 dataset's limited racial diversity created potential bias issues. While we acknowledged this limitation, we prioritized having a working system over perfect fairness for this educational project.

Model Complexity: The LookALikeFinder proved more challenging than anticipated due to the large number of celebrity classes and image variations. We simplified the approach to ensure timely delivery.

Infrastructure: Our initial DevOps setup was unstable, leading to multiple VM rebuilds. We eventually stabilized the deployment with better automation and monitoring.

B. Successes

Complete Pipeline: We successfully built and deployed an end-to-end system from mobile app to ML models.

Code Quality: Implemented clean architecture, proper testing, and version control for both data and models.

User Experience: Created a functional iOS app with camera integration and smooth navigation.

VI. Conclusion

We successfully delivered a functional facial analysis system that meets our core requirements. While there are limitations in model performance and dataset diversity, the project demonstrates a complete machine learning pipeline from research to deployment.

Key achievements include:

- Two trained ML models for facial analysis
- Deployed backend API with user authentication
- Functional iOS mobile application
- Robust MLops infrastructure with Mlflow and DVC

A. Future Work

We would focus on improving model accuracy, addressing dataset bias, and implementing the planned reinforcement learning component for adaptive celebrity matching.

VII. Work Distribution

Aleksandr: Backend infrastructure, deployment, and MLops setup.

Makar: Model development, training pipelines, and dataset management.

Ivan: Mobile app development, UI/UX design, and integration testing.

References

[1] Serengil, S. I. (2019). "Celebrity Look-Alike Face Recognition with Deep Learning in Keras." https://sefiks.com/2019/05/05/celebrity-look-alike-face-recognition-with-deep-learning-in-keras/
[2] "The celebrity lookalike makeup trend is everywhere. Here's why it's a problem." Mamamia. https://www.mamamia.com.au/celebrity-look-alike-makeup-trend-tiktok/
[3] Liang, L., Lin, L., Jin, L., Xie, D., & Li, M. (2018). "SCUT-FBP5500: A diverse benchmark dataset for multi-paradigm facial beauty prediction." In 2018 24th International Conference on Pattern Recognition (ICPR).
[4] Parkhi, O. M., Vedaldi, A., & Zisserman, A. (2015). "Deep face recognition." In BMVC.
[5] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). "FaceNet: A unified embedding for face recognition and clustering." In CVPR.
[6] "Gradient: Celebrity Face Match." App Store. https://apps.apple.com/us/app/gradient-celebrity-face-match/
[7] "FaceApp: AI Face Editor." https://www.faceapp.com/
[8] "MLflow: A Platform for the Machine Learning Lifecycle." https://mlflow.org/
[9] "DVC: Data Version Control." https://dvc.org/
[10] "FastAPI: Modern Python Web Framework." https://fastapi.tiangolo.com/
[11] Paszke, A., et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In NeurIPS.
[12] "Flutter: Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase." https://flutter.dev/
[13] Jones, M., Bradley, J., & Sakimura, N. (2015). "JSON Web Token (JWT)." RFC 7519.
[14] T. Sano (2025). "Male and female facial attractiveness prediction: An image-based approach using convolutional neural network-based models." doi:10.1016/j.fsigen.2025.100042
[15] "Docker: Empowering App Development for Developers." https://www.docker.com/