Facial Attractiveness Level Estimation CS559 Assignment

Mayasah Lami
Department of Computer Engineering
Bilkent University
m.lami@bilkent.edu.tr

Utku Boran Torun
Department of Computer Engineering
Bilkent University
boran.torun@bilkent.edu.tr

I. Introduction

Facial attractiveness estimation is a regression task with applications in recommendation and HCI. We study CNN design choices on 80×80 facial crops from SCUT-FBP5500 (train 3556 / val 894 / test 896; Fig. 1). We vary depth, normalization, regularization, and optimization across 25 runs and reach 0.78 test MAE.

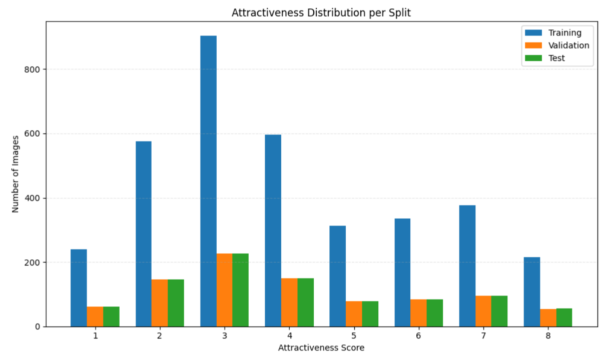


Fig. 1. Dataset distribution

II. Methodology

A. Data Loading and Preprocessing

Images (JPEG) are decoded, resized to 80×80, and normalized to $[0, 1]$. During training, we apply light augmentation (random horizontal flip; mild brightness/contrast/saturation jitter) to preserve facial structure. Harsh geometric transforms were avoided to preserve facial structure.

B. Baseline Architecture

Our baseline comprises:

- **Conv blocks:** 3×3 convolution + ReLU + 2×2 max pooling
- **Global average pooling:** Spatial reduction
- **FC layer:** 128 units with ReLU
- **Output:** Single linear unit for regression

C. Training Configuration

All models used Adam optimizer [1], MAE loss, batch size 64, and early stopping (patience 6 on validation MAE).

D. Experimental Design

We validated the pipeline locally and trained final models on a GPU server (TensorFlow 2.20.0). [1].

We systematically varied:

- **Depth**: 3–5 convolutional blocks
- **Filters**: Progressive configurations (e.g., [32, 64, 128, 256])
- **Batch normalization**: Enabled/disabled
- **L2 regularization**: 1×10^{-5} to 1×10^{-4}
- **Dropout**: 0.1–0.3
- **Data augmentation**: With/without
- **Hyperparameters**: Learning rate 1×10^{-3}, batch size 64 (fixed unless noted)

E. Experimental Flow

We followed a systematic bottom-up approach:

1) **Baseline:** 3-layer CNN without regularization achieved test MAE ≈ 1.02 for both Glorot and Gaussian initialization (50 epochs).
2) **Regularization:** Data augmentation provided minimal improvement (0.816 vs 0.811 MAE). Higher L2 (10^{-4}) and dropout (0.3) slowed convergence without significant gains.
3) **Architecture:** Deeper 5-layer models with batch normalization improved stability; removing BN caused overfitting and higher variance.
4) **Fine-tuning:** Optimizing dropout (0.1–0.25) and learning rate yielded stable convergence at 60–70 epochs with validation MAE ≈ 0.92 and test MAE ≈ 0.81.

III. Network Architecture

The final architecture was selected after iterative experimentation with depth, normalization schemes, initialization strategies, and regularization techniques. The complete model is illustrated in Fig. 2.

A. Overall Structure

The proposed CNN consists of four convolutional blocks followed by a regression head. Each block contains a 3×3 convolutional layer, batch normalization, a ReLU activation, 2×2 max pooling, and dropout with probability 0.2. As the

[1]https://github.com/tensorflow/tensorflow/tree/r2.20

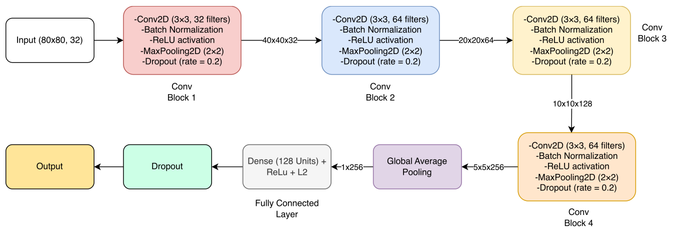


Fig. 2. Architecture of our Model

depth increases across the network, the number of feature filters grows from 32 to 256, while the spatial resolution decreases from $80{\times}80$ to $5{\times}5$. This structure enables the network to capture low-level facial textures in the early layers and higher-level semantic features in the deeper layers. After the convolutional feature extractor, a Global Average Pooling (GAP) layer compresses the final activation map into a 256-dimensional embedding, which serves as input to the regression head.

B. *Convolutional Feature Extractor*

As depicted in Fig. 2, the convolutional backbone contains four stages:

- **Block 1:** Conv2D–32, BatchNorm, ReLU, MaxPool, Dropout.
- **Block 2:** Conv2D–64, BatchNorm, ReLU, MaxPool, Dropout.
- **Block 3:** Conv2D–128, BatchNorm, ReLU, MaxPool, Dropout.
- **Block 4:** Conv2D–256, BatchNorm, ReLU, MaxPool, Dropout.

The corresponding feature map dimensions across these blocks are $80{\times}80{\times}32$, $40{\times}40{\times}64$, $20{\times}20{\times}128$, and $5{\times}5{\times}256$, respectively. Batch normalization improves stability and accelerates convergence, while dropout and max pooling contribute to regularization and spatial downsampling. We experimented with strided convolutions but observed more stable behavior using explicit pooling operations for this regression problem.

C. *Regression Head*

Following the conv blocks, we use Global Average Pooling (GAP) to obtain a 256-dimensional vector, then a 128-unit ReLU layer with dropout 0.2 and a linear Dense(1) unit for regression. We report MAE (rounded and raw).

IV. Baseline Architecture: 3-Layer CNN

A. *Weight Initialization*

We compared Xavier (Glorot) and Gaussian initialization on a 3-layer CNN [32, 64, 128] without batch normalization or regularization.

Xavier initialization [2] samples weights from a uniform distribution scaled by layer dimensions to preserve activation variance:

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right) \tag{1}$$

Gaussian initialization uses fixed standard deviation ($\sigma = 0.05$):

$$W \sim \mathcal{N}(0, \sigma^2) \tag{2}$$

TABLE I
Initialization Comparison

Initialization	Val MAE
Glorot	1.115
Gaussian	1.107

As the difference was not huge, we decided to continue experimenting with both.

B. *Adding Batch Normalization*

Batch normalization (BN) normalizes layer inputs, reducing internal covariate shift and allowing higher learning rates [3]. It often improves training stability and final performance.

Setup: 3-layer CNN with Glorot and Gaussian initialization, adding BN after each convolutional layer, still no L2 or dropout (l2=0.0, dropout=0.0).

TABLE II
Effect of Batch Normalization

Config	Val MAE (Gaussian)	Epoch	Val MAE (Xavier)	Epoch
No BN	1.107	47	1.115	49
With BN (r1)	1.12	36	1.037	25
With BN (r2)	1.02	35	0.988	35

Observation: BN shows variable but generally positive effects, especially on Xavier initialization. Best run achieves 0.926 test MAE (6.5% improvement). BN accelerates convergence significantly (epoch 25-35 vs 49). Variability between runs suggests sensitivity to random seed.

We then included BN in all subsequent architectures, and decided to test using Xavier initialization as it yielded better validation MAE and converged faster.

C. *Exploring L2 Regularization*

L2 weight regularization penalizes large weights, potentially reducing overfitting [4]. We test multiple regularization strengths to find optimal balance.

Setup: 3-layer CNN with BN and Glorot initialization, testing L2 penalties: 0.0, 10^{-5}, 10^{-4}, 10^{-3}.

TABLE III
L2 Regularization on 3-Layer CNN with BN

L2 Weight	Val MAE	Epoch
0.0	0.988	35
10^{-5}	1.024	26
10^{-4}	0.985	42
10^{-3}	1.041	34

Observation: Surprisingly, L2 regularization degrades performance on this 3-layer architecture. The unregularized model achieves best test MAE (0.926). Increasing regularization strength progressively worsens results, with 10^{-3} nearly returning to baseline performance (0.994).

Hypothesis: The 3-layer architecture is capacity-limited. It lacks sufficient parameters to model the task while simultaneously satisfying weight penalty constraints. Regularization is counterproductive when model capacity is already a bottleneck.

Decision: For 3-layer networks, avoid heavy regularization. This motivated exploring deeper architectures with greater capacity.

D. Dropout Regularization on 3-Layer Network

Dropout provides implicit ensemble learning by randomly dropping units during training [5]. We test whether dropout complements BN or improves upon light L2 regularization.

Setup: 3-layer CNN with BN, L2=10^{-5}, testing dropout rates: 0.0, 0.1, 0.2, 0.3.

TABLE IV
DROPOUT EFFECT ON 3-LAYER CNN

Dropout	Test MAE	Val MAE	Loss
0.0	0.933	1.024	MAE
0.2	1.015	1.057	MAE
0.2	1.058	1.141	Huber
0.3	1.120	1.190	MAE

Observation: Dropout substantially degrades performance. Validation MAE increases from 1.024 to 1.057 with dropout=0.2, and further to 1.190 with dropout=0.3. Huber loss performs on similar level.

Interpretation: This reinforces our capacity-limitation hypothesis. Dropout reduces effective model capacity during training, exacerbating the bottleneck. For this shallow architecture, dropout is counterproductive.

Decision: Move to deeper architecture before further exploring regularization techniques.

E. Scaling to 4-Layer Architecture

Given that 3-layer networks suffer from capacity limitations, we hypothesize that a deeper architecture with more filters can better leverage regularization techniques while achieving superior baseline performance.

Setup: 4-layer CNN with filters [32, 64, 128, 256], BN, Glorot initialization. We immediately test with regularization (L2 and dropout) given our understanding that this architecture should have capacity to spare.

TABLE V
4-LAYER CNN WITH REGULARIZATION

L2 / Dropout	Val MAE	Epoch
10^{-4} / 0.2 (r1)	0.919	50
10^{-4} / 0.2 (r2)	0.943	39
10^{-5} / 0.2	**0.916**	63
10^{-5} / 0.1	0.920	57

The 4-layer architecture achieves 0.916 validation MAE compared to 1.024 for the best 3-layer model, a 10.5% error reduction. This validates our capacity hypothesis.

We tried the best configuration on a 5-conv network (filters [32, 64, 128, 256, 512]), however, the test MAE shot up from 0.811 to 0.846. We therefore chose to continue with the 4-conv configuration as it proved most balanced.

Regularization now helps: lighter L2 (10^{-5}) outperforms heavier regularization (10^{-4}), and moderate dropout (0.1-0.2) is beneficial. The optimal configuration uses L2=10^{-5} with dropout=0.2.

F. Loss Function Comparison

MAE yields slightly lower test error; Huber converges faster. We use MAE for final results.

a) Mean Average Error (MAE): loss function calculates the average size of the differences between predicted and actual values without taking into account their direction [6], calculated as follows:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3}$$

where y_i is the true attractiveness label, $\hat{y}_i$ is the predicted attractiveness score, and N is the total number of samples.

b) Mean Squared Error (MSE): is sensitive to larger errors due to its squaring operation, which amplifies the impact of outliers. This loss function is particularly effective in scenarios where addressing significant deviations is a priority, as it penalizes larger errors more heavily than smaller ones. It is mathematically defined as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4}$$

c) Huber Loss: integrates the advantages of Mean Absolute Error (MAE) and Mean Squared Error (MSE) by exhibiting MSE characteristics for minor discrepancies and MAE characteristics for significant ones, making it robust to outliers [7]. The Huber Loss is defined as:

$$L_\delta(y_i, \hat{y}_i) = \begin{cases} \frac{1}{2}\left(y_i - \hat{y}_i\right)^2 & \text{if } |y_i - \hat{y}_i| \le \delta, \\ \delta\,|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{if } |y_i - \hat{y}_i| > \delta. \end{cases} \tag{5}$$

where δ is a threshold parameter that controls the transition between MAE and MSE behavior.

Setup: 4-layer CNN with optimal regularization (BN, L2=10^{-5}, dropout=0.2), comparing MAE vs Huber loss.

TABLE VI
LOSS FUNCTION COMPARISON

Loss	Val MAE	Epoch
MAE	0.916	63
Huber	0.917	42

MAE loss achieves slightly better validation performance (0.01 difference), though the difference is modest. Huber loss converges 33% faster (42 vs 63 epochs), suggesting smoother optimization. We used MAE loss for best final performance.

G. Data Augmentation

Data augmentation can act as implicit regularization by artificially expanding the training set. We test whether augmentation further improves our well-regularized model.

Setup: 4-layer CNN with BN, L2=10^{-5}, dropout=0.2, with/without data augmentation (random flips, shifts, rotations).

TABLE VII
DATA AUGMENTATION IMPACT

Augmentation	Val MAE	Epoch
No	0.916	63
Yes	0.932	43

Observation: Data augmentation slightly degrades performance despite faster convergence. This is counterintuitive but explainable.

Hypothesis: Either (1) augmentation introduces train-test distribution mismatch for facial attractiveness (e.g., horizontally flipped faces may have different perceived attractiveness), or (2) existing regularization (dropout + L2) is already sufficient, making additional implicit regularization redundant.

Decision: Do not use data augmentation for final model.

H. Hyperparameter Sensitivity

1) Learning Rate: We briefly test reducing learning rate from 0.001 to 0.0005 with augmentation.

TABLE VIII
LEARNING RATE SENSITIVITY

Learning Rate	Val MAE	Epoch
0.001	0.932	43
0.0005	0.978	36

Observation: Halving the learning rate substantially degrades performance. The original learning rate of 0.001 is well-suited for this architecture and optimizer configuration.

2) Filter Capacity: We test whether doubling filter counts improves performance.

TABLE IX
FILTER CAPACITY EXPLORATION

Filters	Val MAE	Aug
[32,64,128,256]	0.916	No
[64,128,256,512]	0.946	Yes

Observation: Doubling filter counts degrades performance (0.898 vs 0.811), likely due to overfitting despite regularization. The original configuration provides optimal capacity.

I. Fine-Tuning

We fine-tuned our best model (lowest validation MAE of 0.916, also lowest test MAE of 0.81) with 10× reduced learning rate.

Configuration:

- Base model: 4 layers, [32,64,128,256], BN, L2=1e-5, dropout=0.2
- Fine-tuning: LR=1e-4 (from 1e-3), 150 epochs, patience=20

Results: Val MAE = 0.78, Test MAE = **0.78**

Final Improvement: 0.81 → 0.78 (4% reduction)

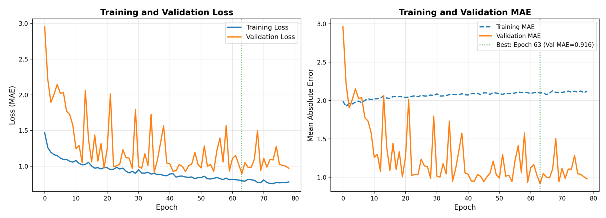
The reduced learning rate allowed precise weight adjustments, escaping a shallow local minimum. Training curves showed gradual improvement over 63 epochs before early stopping, demonstrating the model's capacity for further optimization beyond initial training.

J. Final Model Performance

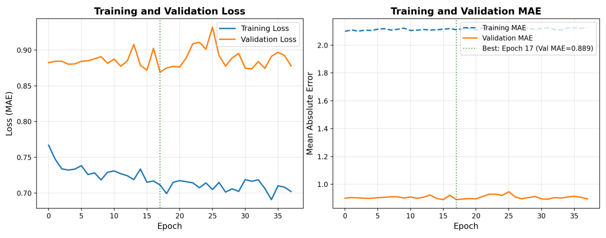
Our best model achieved:

- **Test MAE (rounded): 0.78**
- **Test MAE (raw): 0.80**
- RMSE: 1.05

The loss graph of our best performance models can be seen in Figure 3.



(a) Best model chosen for fine-tuning



(b) Fine-tuned model

Fig. 3. The loss function graphs for the best model chosen for fine-tuning, as well as the fine-tuned model

V. RESULTS

A. Success Cases

Figure 4 shows four examples with near-perfect predictions (absolute errors 0.001–0.012). These successes span multiple demographics, demonstrating robust feature extraction across diverse appearances. Common factors include neutral expressions, frontal pose, minimal occlusion, and uniform lighting; conditions that allow the model to exploit learned representations effectively.

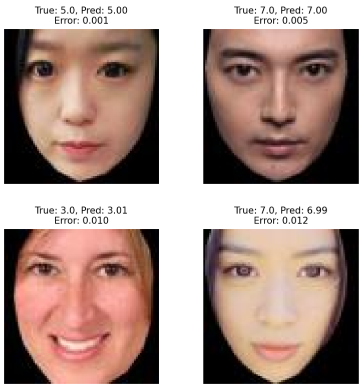


Fig. 4. Best predictions from test set (errors < 0.02)

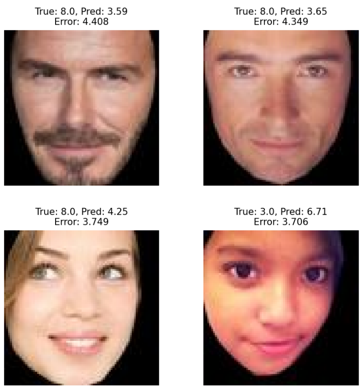


Fig. 5. Worst predictions from test set (errors 3.7–4.4)

B. Failure Cases

Figure 5 presents the highest-error cases, with mispredictions of 3.7–4.4 points. Failures consistently involve atypical poses, strong expressions, upward gaze, or uneven lighting that degrades feature extraction. Some faces exhibit attribute combinations underrepresented in training, suggesting limited generalization to rare demographic patterns.

These analyses reveal that the model performs reliably under controlled conditions but remains sensitive to pose variation, lighting imbalance, and distributional shifts.

VI. Discussion

A. Key Findings

BN is critical: largest single gain and faster convergence. **Right-sized capacity:** 4 conv blocks outperform 3 (too small) and wider/deeper variants (overfit). **Light regularization + fine-tuning:** L2=10^{-5}, dropout 0.2, then LR decay yields the final 0.78 MAE.

B. Failed Experiments

Strided conv in place of max pooling and wider backbones overfit or underperformed (e.g., MAE 1.00 and 0.90 respectively). Transfer learning with VGG16 at 80×80 performed poorly (Table X), reflecting a mismatch in input resolution and task (classification vs. scalar regression). These results reinforce the choice of a lightweight, task-specific CNN.

TABLE X
Transfer Learning vs Custom Architecture

Configuration	Test MAE
Our custom CNN	**0.811**
VGG16 frozen	1.489
VGG16 + block5 trainable	1.486

The VGG16 models achieved test MAE of approximately 1.49—83% worse than our custom architecture. This substantial failure stems from fundamental mismatches: VGG16's features are optimized for 1000-class object recognition rather than continuous regression, our 80×80 inputs are far smaller than VGG16's expected 224×224 resolution, and the model's depth over-parameterizes this relatively simple dataset. Fine-tuning block5 provided negligible improvement (0.003 MAE), confirming that the core issue is architectural mismatch rather than insufficient adaptation.

These failures demonstrate that neither depth nor pre-trained representations guarantee improvement. Success requires matching architecture complexity to input resolution and dataset characteristics. For our 80×80 facial images and moderate-sized dataset, a purpose-built lightweight CNN substantially outperforms both over-parameterized and transfer learning approaches.

VII. Conclusion

This work systematically investigated CNN architectures and training techniques for facial attractiveness estimation. Through 25 experiments, we improved from a baseline of 1.03 MAE to a final result of 0.78 MAE (24% improvement). Key insights include the critical role of batch normalization, the effectiveness of minimal L2 regularization, the importance of appropriate architecture depth for small images, and the value of fine-tuning for final performance refinement.

Our experiments demonstrate that systematic, progressive experimentation—building complexity incrementally and analyzing each component's contribution—enables both strong performance and deep understanding of model behavior. The failed experiments (strided convolutions, over-wide networks) proved as instructive as successes, revealing the constraints imposed by image resolution and dataset size.

Future work could explore ensemble methods, attention mechanisms, or multi-task learning approaches to further improve performance.

References

[1] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[2] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.

[4] C. Cortes, M. Mohri, and A. Rostamizadeh, “L2 regularization for learning kernels,” *arXiv preprint arXiv:1205.2653*, 2012.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[6] J. Qi, J. Du, S. M. Siniscalchi, X. Ma, and C.-H. Lee, “On mean absolute error for deep neural network based vector-to-vector regression,” *IEEE Signal Processing Letters*, vol. 27, pp. 1485–1489, 2020.

[7] P. J. Huber, “Robust estimation of a location parameter,” in *Breakthroughs in statistics: Methodology and distribution*. Springer, 1992, pp. 492–518.