CS559 Homework
Estimation of Facial Attractiveness Level using TensorFlow

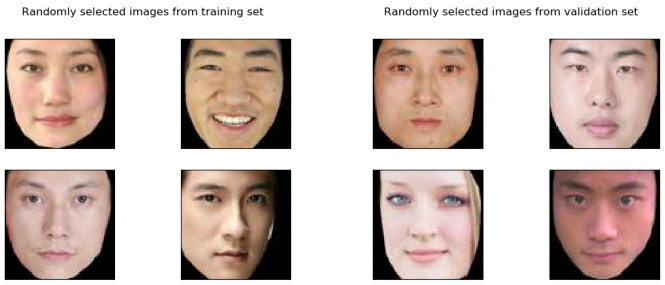
Selim Furkan Tekin
21501391

Fatih İlhan
21401801

***Abstract*—In this homework, we design and train a deep neural network for facial attractiveness level estimation. In particular, we develop an architecture based on convolution, pooling, activation and fully-connected layers. We analyze the performance of our model over a preprocessed facial image dataset with attractiveness level annotations. In this report, we describe our architecture, training procedure and obtained results on the given dataset. The architecture can reach 0.42 and 0.50 mean absolute error in validation and test sets respectively.**

I. Data Analysis and Preprocessing

In this section, we give the details of our analysis on the given dataset before starting to work on model design. Data analysis is a crucial step before designing a deep neural network since the statistics of data can significantly affect the design choices. To this end, we perform certain analyses on training and validation sets of the given SUCT dataset [1]. First, we display some randomly selected images and the mean of all images in the datasets. In the following figure, we display randomly selected images from the dataset:

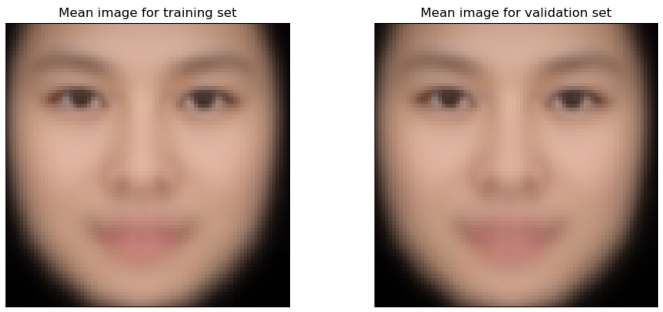


(a) Training (b) Validation

Fig. 1: Randomly selected images from the given dataset

Thankfully, the dataset is preprocessed. The facial images are cropped properly, and irrelevant regions are masked. Their rotational alignment is also handled. There is no significant pose or illumination difference between samples. In addition, preprocessing is conducted for both training and validation sets. Therefore, we do not need to perform any preprocessing further except scaling the images into values between 0 and 1.

In addition, we analyze the mean and variance images of the given dataset. We display the mean images of training and validation sets to see if there is any significant difference between training and validation sets. These images are displayed in Fig. 2. The mean images of training and validation sets are almost the same visually. Therefore, the feature space distribution difference between training and validation sets is not significant. This fact strengthens our motivation to apply deep learning for this task, since the learned model in the training set will be able to represent the validation set, and hopefully test set.



(a) Training (b) Validation

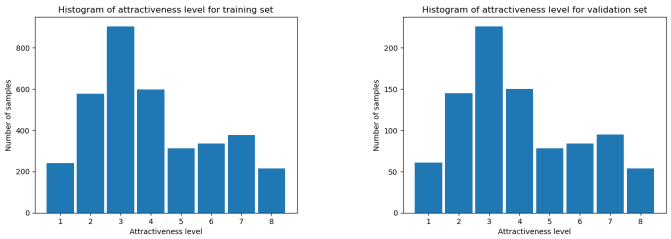
Fig. 2: Mean images of the given dataset

We also analyze the distribution of attractiveness levels using a histogram. In Fig. 3, we display the number of samples from each level in training and validation sets. We observe that the imbalance between levels is significant. In particular, lower attractiveness levels are more common than higher attractiveness levels. However, this distribution stays same in the validation set, which shows that the dataset is splitted into sets considering the level distribution.

II. Design and Training Procedure

In this section, we describe the design and training procedure of our deep convolutional network. Before starting random hyperparameter search and finetuning, we first determine the loss function, optimizer, model structure, and layer types. Since we do not have enough time and computational resources to try all possible design possibilities, we need to give certain main design choices.

First, we determine our loss function, optimizer, and the initialization method. In the homework description, it is mentioned that the evaluation will be conducted in terms of mean



(a) Training (b) Validation

Fig. 3: Histogram of attractiveness levels

Layer	Filter Size	Stride	# Filters	Output Size
conv_1	5	3	16	?x?x16
conv_2	5	1	16	?x?x16
max_pool_1	5	1	-	?x?x16
conv_3	3	3	32	?x?x32
conv_4	3	1	32	?x?x32
max_pool_2	3	1	-	9x9x32
flatten	-	-	-	2592
dense_1	-	-	-	256
dense_2	-	-	-	1

TABLE I: Initial model structure

Layer	Filter Size	Stride	# Filters	Output Size
conv_1	5	3	8	?x?8
conv_2	5	1	16	?x?x16
max_pool_1	5	2	-	?x?x16
conv_3	3	1	16	?x?16
conv_4	3	1	16	?x?16
max_pool_2	3	2	-	7x7x16
flatten	-	-	-	784
dense_1	-	-	-	64
dense_2	-	-	-	1

TABLE II: Final model structure

absolute error. Since this is a regression task in which we want greater errors to be penalized heavier, we use mean squared loss while optimization. We train our models and perform early stopping based on this loss. However, we evaluate and select the models based on their mean absolute errors in the validation set. As requested in the homework description, we stick with Adam optimizer, which has shown fast convergence in almost all of the convolutional neural network designs in the literature. Although there are several variants of it, we use Adam optimizer with its default form, and default decay rates in Tensorflow. For the initial set of experiments, we utilize Xavier initialization to prevent the weight distribution of layers from disrupting. Since we expect this method to perform better than random initialization through keeping the mean and variances of weights in reasonable ranges without saturation, it will expose the potential of models more successfully for the initial experiments. However, we also try random initialization at the last finetuning stage since the performance differences in initial experiments were not very significant.

Once determining the objective function and initialization, we start to design our model. Instead of trying numerous models with different number of layers/filters, and various convolutional and pooling filter sizes/strides, we follow a systematic procedure. First, we aim to overfit to the training set with a reasonably small model. Since we plan to regularize and modify it later, overfitting at this stage is just a proof of validity of the general structure of our model. After a fast manual search, we have ended with a model with the structure described in Table I.

This model consists of 4 convolutional layers, 2 max-pooling layers, and 2 dense layers. For convolutional layers, we use the "SAME" option instead of "VALID" to include padding and prevent the outermost pixels to drop after each convolution. In additio, we use leaky relu with a slope of $\alpha = 0.01$ after each convolutional layer and the first dense layer. We use the standard relu function at the output of our model. By using leaky relu, we aim to continue gradient flow even if there are negative values at the output of the layers. We also apply batch normalization after activations of convolutional layers. Without a fine search, we set the learning rate to $2e-3$, and mini-batch size of 32. We observed that training was very noisy if the mini-batch size is small and slow if it is very high. Since we want to overfit at this step, we do not any regularization. In this setup, the training MSE loss drops to 0.18 and 0.11 around 100th and 300th.

After observing that the proposed structure given in Table I is suitable for the given task, we have added early stopping which stops training if the mean squared loss in the validation set does not decrease 5 consecutive epochs. With this setup, our rounded mean absolute error was 0.79. To see the effect of batch normalization, we have run the same setup without it and as expected, we have observed a more noisy loss decay during the training and the validation score was around 0.88. Therefore, we apply batch normalization for all experiments after this point.

Until this stage, our only regularization technique was early stopping. However since our dataset is not large and the training error is much lower than validation loss during training, we have added L2 regularization ($\lambda = 1e-3$) for all layers and dropout (0.1) for the dense layers. After stronger regularization, our validation loss has decreased to 0.74. Then, we have tried slightly deeper and wider versions of this model by increasing the number of filters and layers in our network. We have tried several variants with additional two layers and/or doubled number of filters however, we could not obtain any significant improvement. Moreover, the network has become unstable and validation loss plots were noisier.

Since we could not obtain any performance gain, we have tried a simpler version with fewer filters and pooling strides as described in Table II. This model was able to drop the validation loss to 0.69 with the same optimization and regularization parameters. Decreasing the number of layers was not helpful, therefore we select this configuration as our final model structure. To observe the effect of our choices over the performance, we provide the results in Table III. Note that these models are not fine-tuned. In the following section, we describe the hyperparameter search and finetuning procedure

for learning rate, batch size, regularization parameters, and initialization method.

Best Model	No-BN	No-Xavier	No-Reg	Complex	Simple
0.69	0.88	0.71	0.74	0.71-0.79	0.74-0.77

TABLE III: Rounded MAE in the validation set for the first experiment stage. Best configuration, no batch normalization, random initialization, no regularization, more complex network and simpler network results.

III. Results

In the previous section, we have determined the structure of our model, objective function and the optimization method. In this section, we show the final results in the validation and test sets after finetuning. Furthermore, we show the effects of parameters on the performance by comparing their results and monitoring the learning curves.

A. Hyperparameter Search and Finetuning

In the first step of parameter search, we have decided on a search range for each parameter. Thus, we have monitored the decrease in MSE loss in the validation set to see which parameters work better and consistently. For example, for the learning rate we have selected four possible values in log base. We decreased the upper bound of the search range, if the loss oscillates too much, and increased the lower bound of the search range if the loss decreases too slowly.

For the regularization parameters, we have visually monitored the loss curve and investigate the gap between the validation and training curves. If there is a large gap, we strengthen our regularization by increasing λ parameter in $L2$ regularization and drop-out rate. Batch normalization has provided better results along with stability and robustness in parameter search, thus we have applied batch regularization between every convolutional layer.

After determining the search ranges, we have conducted a grid search to obtain different combinations of parameters. Then we performed experiments for each combination after shuffling the configurations randomly. Early stopping has increased the speed of experiments and allowed us to perform more experiments in a limited time.

Fig. 4 shows the learning curve of the final model. Table IV shows the parameters and the results for the final model. Among 240 experiment results, the configuration with the smallest validation $l1$ loss is selected as the best model configuration. We have obtained $\mathbf{0.50}$ MAE on the test set with the best model. The learning curve shows that learning rate is tuned well since it has a sharp initial decrease and reaches a stable and low loss value. In addition, the gap between train and validation losses is not significant, since we have used early stopping so that the model is stopped when validation loss had stopped decreasing. Fig. 5 shows sample output predictions taken from validation set.

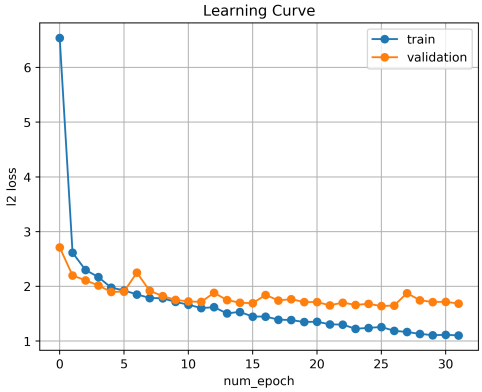


Fig. 4: Learning curve of final model

Fig. 5: Predictions for the samples taken from the test set. Sample labeled as 8 could be biased since sample is a celebrity. We also observe that similing faces have higher scores, so the first image is predicted as more attractive and the last image is predicted as less atractive.

TABLE IV: Parameter Values for Final Model

Parameter Name	Parameter Value
Batch size	64
Learn rate	0.0003
optimizer	Adam
Init type	Random
Loss type	l2
Alpha	0.01
Batch Regularization	True
Lambda	0.001
Drop-out Prob	0.1
Stop tolerance	5
Validation Loss	**0.42**

B. Parameter Effects on the Performance

1) Batch Normalization: After every convolutional layer, we have implemented a batch normalization layer. This layer behaves differently during training and prediction stages. In

training, activations are normalized with the mean, μ and variances, σ^2 of the mini batch samples. Then, resulting activations are scaled by γ and shifted with an offset β.

$$\mathbf{x} = \frac{\gamma(\mathbf{x} - \mu)}{\sigma} + \beta \tag{1}$$

Scale and offset parameters are trained during training. In addition, mean and variance parameters are updated by exponential moving average, which averages with a decay rate of 0.5. Mean and variance updates are not applied during test, and the averaged values from training are used. Fig. 6 shows the effect of batch normalization. Batch normalization decreases the dependence on initialization. Furthermore, model converges faster to a lower loss value. This shows the improvement in gradient flow through the network.

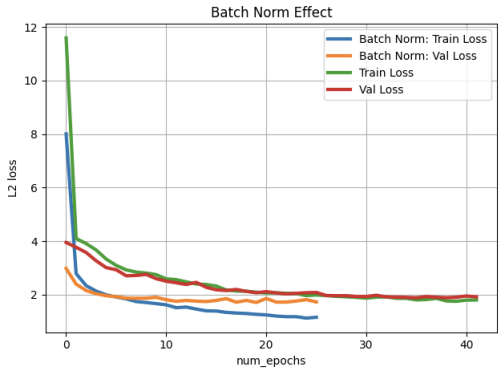


Fig. 6: Effect of batch normalization is shown with learning curves.

2) Weight Initialization: We have investigated the parameter initialization effect by monitoring the first few epoch losses. We observed that Xavier initialization results in a lower loss compared to the random initialization. Fig. 7 shows the learning curves of the experiments with the best configurations, but the initialization methods are different. In both experiments, batch normalization is turned off to see the effect of different initialization more explicitly.

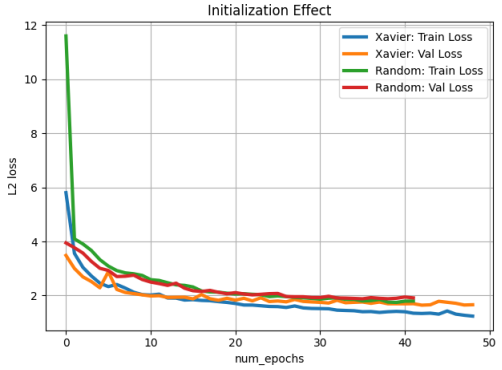


Fig. 7: Effect of initialization is shown with learning curves.

C. Regularization Effect

We have investigated the effect of regularization on learning by monitoring the loss curves as in the previous sections. To this end, we compare two experiment results with two different configurations. One configuration has exactly the same paremeters of best model, and other experiment lacks regularization i.e no batch normalization, no drop-out and no L2 regularization. This combination can be seen on Figure 8. We observed that with regularization, model reached lower loss with few epochs. However, the fact that our model is not deep decreases the need for regularization. So, the regularization does not drastically improve the validation performance although it still helps to prevent overfitting after some epochs. In particular, dropout helps to regularize the last dense layers.

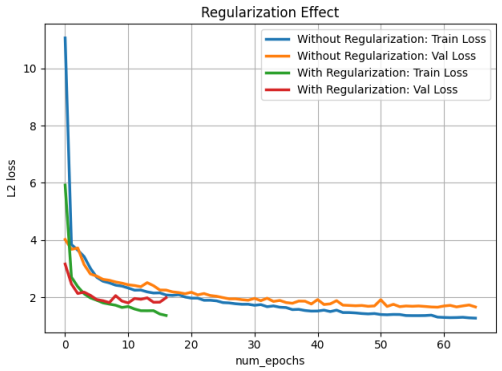


Fig. 8: Effect of regularization is shown with learning curves.

IV. Conclusion

In this work, a deep learning model is implemented to estimate facial attractiveness. First, we have conducted exploratory data analysis on training and validation sets. Then, we have decided on whether we should perform a preprocessing step or not. The images were already cropped according to landmarks, and adjusted in terms of pose and illumination. Thus, no preprocessing is performed except scaling their values to 0-1 range. Second, we have designed our model architecture and decided on the loss type, optimization, and regularization techniques. Then, we have performed hyperparameter search and finetuned our model. Finally, we have investigated the effect of batch normalization, regularization and weight initialization.

We have observed that both batch normalization and regularization ($L2$ and dropout) improves the results. Although we have observed certain improvements in Xavier initialization, our best model uses random initialization. The reason behind this could be the randomness of initialization and the fact that batch normalization decreases the effect of initialization on performance.

Finally, we have accomplished the task with a simple yet efficient model. The architecture could reach 0.42 MAE in the validation and 0.5 MAE in the test sets, which is applicable for a real-life application.

REFERENCES

[1] D. Xie, L. Liang, L. Jin, J. Xu, and M. Li, "Scut-fbp: A benchmark dataset for facial beauty perception," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1821–1826, IEEE, 2015.