Facial Attractiveness Estimation with Regression CS 559: Deep Learning Homework Report

Berat Biçer
Computer Engineering
Bilkent University
Ankara, Turkey
berat.bicer@bilkent.edu.tr

Irmak Türköz
Computer Engineering
Bilkent University
Ankara, Turkey
irmak.turkoz@bilkent.edu.tr

***Index Terms*—Deep Learning, Facial Attractiveness, Regression**

I. Introduction

This report contains the results of the homework and the related discussions. We start of by discussing in detail findings related to individual hyperparameters and design choices, and how they affect validation performance. Then, we provide test results on the network trained using the findings from section 2. We finish the report with a short summary of findings and comments on the performance on test data.

II. Experimental Findings

Unless stated otherwise, term model refers to the convolutional neural network (CNN) as described in Table I which is trained for 10 epochs with learning rate 2.5×10^{-5}, Adam optimizer with default Tensorflow hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$, no dropout or l2-regularization, batch normalization after every convolution, ReLU activations, using mean absolute error (MAE) as train and validation loss, with batch size of 1.

Name	Filter	Stride	Pad	Weights	Bias
Conv1	$4 \times 3 \times 3$	1	1	-	1×4
Maxpool1	2×2	2	0	-	-
Conv2	$8 \times 3 \times 3$	1	1	-	1×8
Maxpool2	2×2	2	0	-	-
Conv3	$16 \times 3 \times 3$	1	1	-	1×16
Maxpool3	2×2	2	0	-	-
Conv4	$16 \times 3 \times 3$	1	1	-	1×16
Maxpool4	2×2	2	0	-	-
Flatten	-	-	-	-	-
Dense1	-	-	-	400×64	1×64
Dense2	-	-	-	64×8	1×8
Output	-	-	-	8×1	1×1

TABLE I: Architecture of the generic model for this report.

A. Model Architecture

B. Initialization

We experimented with two initialization methods, namely Xavier (or Glorot) and random initialization. We trained two instances of a modified version of the network shared in Table I such that convolutional layers (Conv$x$) have $8\times3\times3$, $8\times3\times3$, $16 \times 3 \times 3$, $16 \times 3 \times 3$ filters and 1×8, 1×8, 1×16, 1×16 bias vectors respectively. The models are trained with stochastic gradient descent(SGD) with learning rate 2×10^5 and momentum 0.95 for 10 epochs. The resulting validation loss versus epochs is given in Figure 1.

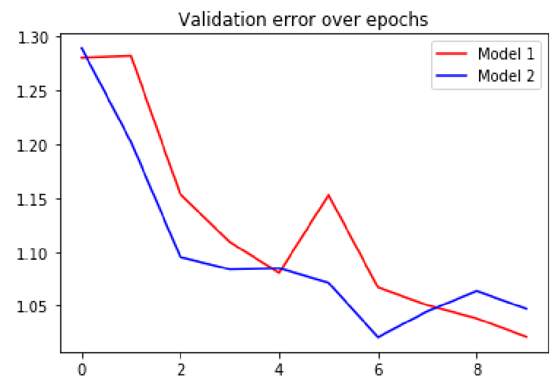

Fig. 1: Plot of validation loss calculated as training progresses over epochs. Red line shows the validation loss for random initialization ($\mu = 0, \sigma = 0.05$), blue line shows validation loss for Xavier initialization.

Notice model 2 (random) has overall lower validation loss than model 1 (Xavier). This can be explained by the model architecture. Xavier initialization seeks to make variance of the outputs of a layer equal to the variance of its inputs and is designed to stabilize networks with sigmoid activations. Since our network has no such layers, random initialization is a better choice than Xavier. This is also supported by other experiments we omitted due to page limit, that is performed under similar conditions with different network architectures that uses ReLU activations.

C. Batch Normalization

Using same model architecture as in Section B with random weight initializations, we trained two instances of the model where in one model we apply batch normalization after every convolution layer and in the other no batch normalization is done. Resulting validation loss versus epochs curve is given in Figure 2.

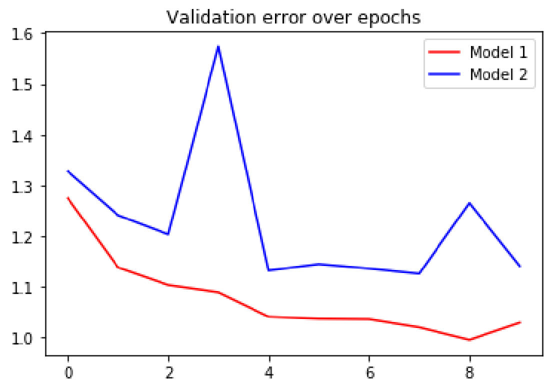

Fig. 2: Plot of validation loss calculated as training progresses over epochs. Red line shows the validation loss with batch normalization, blue line shows validation loss without batch normalization.

Stabilizing effect of batch normalization is clearly visible in Figure 2. Batch normalization applies standard score normalization to weights in a layer to prevent extreme values, resulting in stable gradients. This means loss smoothly improves, as is the case in red curve. If a network lacks batch normalization, gradient updates can become noisy and optimizer fails to find a good optima, which is the case in the blue curve. Similar results are obtained with different networks trained in a similar way, but these results are omitted due to page restrictions.

D. Optimizer

We employed the same network in Part C with batch normalization layers for experiments in this section. However, the network is trained with Adam optimizer and various learning rates. For convenience, default parameters provided by Tensorflow (namely $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$) remains unchanged, since any model trained otherwise performed poorly. We share some of the results for this section in Figure 3.

When compared, we observe that 10^{-4} (magenta curve) converges quickest at around 8th epoch and fluctuates heavily afterwards while others still improve validation loss even after 15th epoch, implying that learning rate for those networks should be larger. Also, notice how quickly network whose loss is shown with magenta line finds a good parameter set at as early as 4th epoch. SGD manages to find such a solution after 8th epoch in Figure 2 and 6th epoch in Figure 1. From omitted experiments, it is also clear that with Adam optimizer networks converge quicker, as expected. Lastly, note that optimal learning rate for this network seem to be somewhere in range $(5 \times 10^{-5}, 10^{-4})$.

E. Network Architecture

In this section, we compare performance of different models with varying architectures trained with Adam optimizer and learning rate 2×10^{-5} for 10 epochs. Model architectures A,B,C, and D are described in Tables II, III, IV, V respectively.

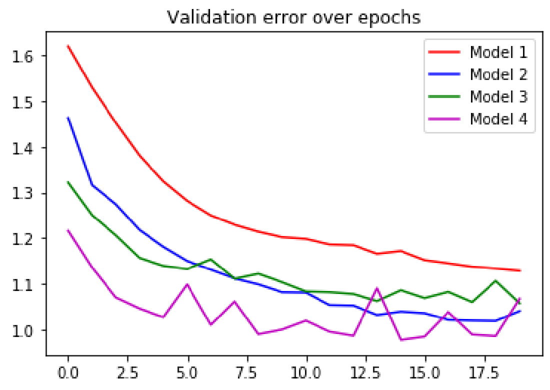

Fig. 3: Plot of validation loss calculated as training progresses over epochs using Adam optimizer. Red line is obtained with learning rate 10^{-5}, blue line with learning rate 2.5×10^{-5}, green line with learning rate 5×10^{-5}, magenta line with learning rate 10^{-4}.

Convolution layers in the models uses 3×3 filters with stride 1, 'same' padding option (Tensorflow keyword stating the padding should be chosen such that output of the filter has same spatial dimensions as the input), and batch normalization is applied after convolution. Max pooling implies 2×2 max pooling with stride 2, effectively halving the input in spatial dimensions. In dense layers, dropout rate is zero. Convolution and dense layers use ReLU activations.

Name	Filter Count	Weights	Bias
Conv1	2	-	2
Maxpool1	-	-	-
Conv2	4	-	4
Maxpool2	-	-	-
Conv3	8	-	8
Maxpool3	-	-	-
Conv4	16	-	16
Maxpool4	-	-	-
Flatten	-	-	-
Dense1	-	400×64	64
Dense2	-	64×8	8
Output	-	8×1	1

TABLE II: Architecture of Model 1

Validation losses given in Figure 4. Model 3 was designed with the assumption that due to the nature of the input (human faces), initial filters lack diversity. Hence, filter diversity increases as the network goes deeper. This assumption is confirmed by the observation that Model 3 has the lowest validation loss. The quality of learned representation is the defining factor in explaining the difference between validation losses observed. Model 1 represents inputs as 400-dimensional vectors before classification whereas model 2 doubles the vectors' length. The fact that model 2 outperforms model 1 implies the model 1 cannot learn a deep representation of inputs. When models 1 and 4 are compared, eventhough these two models produce 400-dimensional features, model 4 outperforms model 1 since model 4 is deeper than model 1.

Name	Filter Count	Weights	Bias
Conv1	2	-	2
Conv2	4	-	4
Maxpool1	-	-	-
Conv3	8	-	8
Conv4	8	-	8
Maxpool2	-	-	-
Conv5	16	-	16
Conv6	16	-	16
Maxpool3	-	-	-
Conv7	32	-	32
Conv8	32	-	32
Maxpool4	-	-	-
Flatten	-	-	-
Dense1	-	800×128	128
Dense2	-	128×16	16
Output	-	16×1	1

TABLE III: Detailed architecture of Model 2

Name	Filter Count	Weights	Bias
Conv1	4	-	4
Maxpool1	-	-	-
Conv2	16	-	16
Maxpool2	-	-	-
Conv3	16	-	16
Maxpool3	-	-	-
Conv4	64	-	64
Maxpool4	-	-	-
Conv5	64	-	64
Maxpool5	-	-	-
Flatten	-	-	-
Dense1	-	256×32	32
Dense2	-	32×4	4
Output	-	4×1	1

TABLE IV: Architecture of Model 3

When model 3 and 4 are compared, we see that model 3 learns a smaller-sized feature vectors, yet this representation captures the nature of the input better than model 4's representation. Difference between models 1 and 3 can be explained by the lack of filter diversity in model 1: model 3 starts with a larger base and exponentially grows in filter size while learning a more compact representation. Model 1 follows a similar pattern yet the growth is slower. This implies that the growth pattern of model 3 is compatible with the data, and supports the assumption behind model 3. It is likely that model 3 can be improved by increasing the filter counts as 8,32,128,etc. which can be done in the future.

Name	Filter Count	Weights	Bias
Conv1	2	-	2
Conv2	2	-	2
Maxpool1	-	-	-
Conv3	4	-	4
Conv4	4	-	4
Maxpool2	-	-	-
Conv5	8	-	8
Conv6	8	-	8
Maxpool3	-	-	-
Conv7	16	-	16
Conv8	16	-	16
Maxpool4	-	-	-
Flatten	-	-	-
Dense1	-	400×64	64
Dense2	-	64×8	8
Output	-	8×1	1

TABLE V: Architecture of Model 4

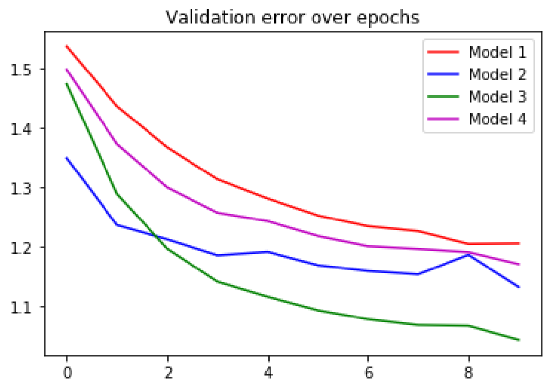

Fig. 4: Plot of validation loss calculated as training progresses over epochs for different architectures.

F. Loss Function

In this section, we trained the model shown in Table I using 4 different loss functions as described below with learning rate 2.5×10^{-5}, however, validation loss is still computed with MAE.

1) Model 1 trained with mean squared error (MSE): $loss_1 = (\frac{1}{n})\sum_{i=1}^{n}(y_i - y_{pi})^2$ where y_i is the label and y_{pi} is the prediction for sample i.
2) Model 2 trained with $loss_2 = (\frac{1}{n})\sum_{i=1}^{n}[(y_i - y_{pi})^2 + |y_i - y_{pi}|]$ where y_i is the label and y_{pi} is the prediction for sample i.
3) Model 3 trained with $loss_3 = (\frac{1}{n})\sum_{i=1}^{n} max((y_i - y_{pi})^2, |y_i - y_{pi}|)$ where y_i is the label and y_{pi} is the prediction for sample i.
4) Model 3 trained with $loss_4 = (\frac{1}{n})\sum_{i=1}^{n} min((y_i - y_{pi})^2, |y_i - y_{pi}|)$ where y_i is the label and y_{pi} is the prediction for sample i.

Loss 1 is the MSE and is susceptible to outliers: since outlier terms cause large losses compared to inliers, the model tends to converge to the mean of the dataset. Loss 2 similar to MSE: for outliers the loss is dominated by the squared term whereas for inliers absolute error determines the loss. This is also true for loss 3. Outliers are penalized by the squared term wheras inliers are penalized by the absolute error. Loss 4, however, follows a different pattern: outliers are penalized by the absolute error where inliers are penalized by quadratic term. This forces the model to partially ignore outlier error and find a good fit prioritizing the inliers. The results from Figure 5 support these claims, since Model 4 (loss 4) has the least validation error after 10 epochs and models 1,2, 3 behave similary.

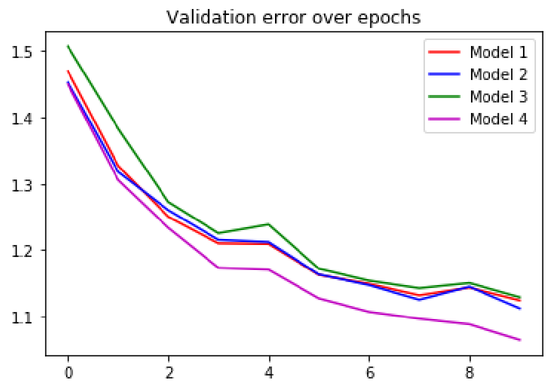

Fig. 5: Plot of validation loss calculated as training progresses over epochs for different loss functions.

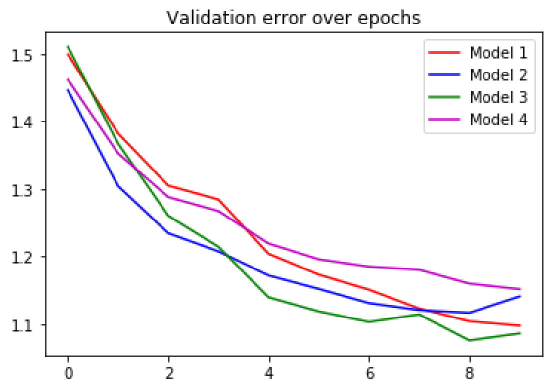

Fig. 6: Plot of validation loss calculated as training progresses over epochs for weight decay values $10^{-5}, 5\times10^{-5}, 10^{-4}, 5\times10^{-4}$ in order.

G. L2 Regularization

In this section, we trained 4 instances of the model described in Table I using 4 unique values of weight decay. Notice that the network has a shallow architecture, hence we expect l2-regularization will be less effective compared to deeper architectures, which is supported by the validation errors obtained in Figure 6 where all networks follow similar patterns. Compared to the base model in Table I, weight decay in fact decreases the performance by forcing the network to keep the weights small rather than learning a good representation.

H. Dropout Regularization

Similar to previous section, we use the same methodology and train 4 models with no weight decay and dropout rates set as 0.5, 0.4, 0.3, 0.2 respectively. The results are given in Figure 7, which are interestingly worse than base model described in Table I. We suspect that our model is too simple for dropout to effectively boost learning: normally, given a complex neural network, dropout forces the model to learn alternative representations for the data. When the

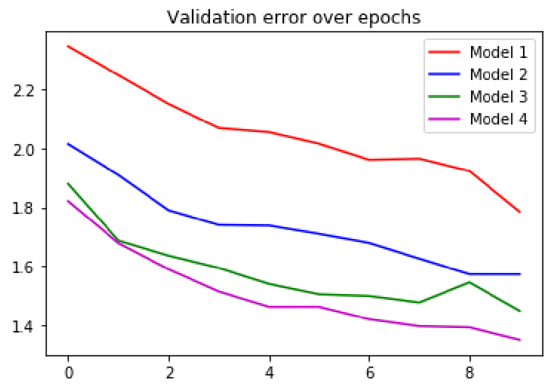

Fig. 7: Plot of validation loss calculated as training progresses over epochs for dropout rates $0.5, 0.4, 0.3, 0.2$ in order.

model is too simple, however, setting some connections to zero might severely debilitate feature extraction since there is no alternate way neurons can communicate with the remaining connections. The fact that default model performs better than any of the models described here is another fact that supports our claim.

I. Batch Size

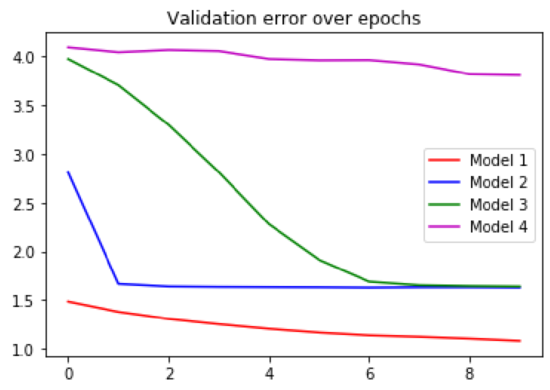

Fig. 8: Plot of validation loss calculated as training progresses over epochs for batch sizes $1, 16, 64, 256$ in order.

Up until this point, every model used batches of size 1 both during training and validation. We trained 4 networks with various batch sizes to see the effect of batch size on performance. The results are in Figure 8. Our initial thought was that there was a bug in the implementation. After spending significant effort into investigating such bugs and failing to detect any, we realized with batch sizes greater than 1 the model learns to predict the mean value of 4. We believe this points at the complexity of the task assigned: facial attractiveness estimation is a hard task; the dataset provided is not diverse enough / has enough samples to allow the network to successfully capture a good representation of the data, and possibly the models we employ are too simple to capture any such representation. Since model 1 (batch size of 1) manages to learn such a representation, the dataset must be of a certain quality; pointing at other factors we described above. It is also possible that as batch size decreases, the number of parameter updates increases, which in return allows the network to find a better local maxima of the underlying function that is different than learning the mean value.

J. Early Stop

Experiments performed throughout this report are designed to overfit by selecting a large number of epochs for training. Spp In Figure 3, we observe the loss curve flattens after 8 epochs for magenta curve and fluctuations begin. As a regularizer, the effect of early stopping is clearly visible here: early stopping before the convergence point allows a somewhat higher generalization by terminating training before such fluctuations begin. When stopped training after 8 epochs, compared to stopping after 14 epochs, it is safe to assume the model has more generalizability.

III. Final Network and Test Results

As per experiments, we trained the final network using the following parameter set and design choices: model 3 from section II.E is trained with Adam optimizer and learning rate 2.5×10^{-5}, random weight initialization, batch normalization, loss function $loss_4$ described in section II.F, weight decay and dropout rate set as 0, and batch size 1, trained for 10 epochs. Early stopping is performed by stopping training after 10 epochs. The resulting validation loss curve is in Figure 9, and the final test loss obtained is $MAE = 0.971$. Sample results from the test dataset are given in Figure 10. These results correspond to samples with first two best and worst test losses in the test dataset. It is interesting that even in case of losses as big as absolute error 4.9, the network manages overall $MAE < 1$, implying performance overall is quite high.

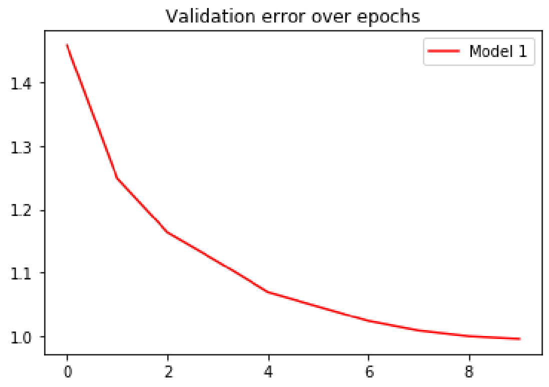

Fig. 9: Final validation loss obtained during training.

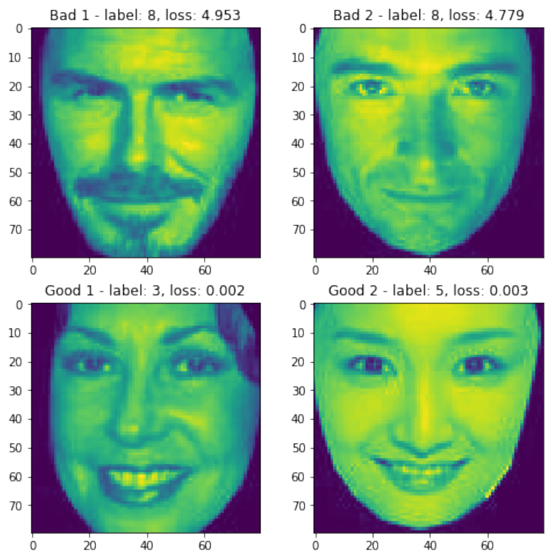
Bad 1 - label: 8, loss: 4.953
Bad 2 - label: 8, loss: 4.779
Good 1 - label: 3, loss: 0.002
Good 2 - label: 5, loss: 0.003

Fig. 10: Sample results from test data. Top row is poor predictions, bottom row are good predictions.