CS 559 Fall 2021 HW Report

Hamza Pehlivan, Hakan Sivuk

I. Introduction

THIS report is intended to serve as a discussion of techniques we tried and motivations behind them. First of all, as it is suggested in the homework file, effects of different methods are examined through several experiments. In Experiments section, comparison of different methods are explained with their motivations, theoretical background and results. After that, evaluation score of the best model according to validation score is stated (on test set). Finally, success and failure cases are shown and possible further improvements are discussed. For all experiments, we used single NVIDIA GeForce GTX 1650 GPU, which has 4 GB memory.

II. Experiments

A. Early Stopping For Preventing Overfitting

Before starting experiments with different settings, early stopping based on validation evaluation score was implemented as a regularization technique. It saves time by stopping the training when the network starts to overfit to the training data. Early stopping algorithm basically checks validation evaluation error at the end of each epoch and compare it with the model that had the smallest validation error. In all of the experiments, we utilized early stopping.

B. Designing The Architecture and Determining The Number of Layers

As a starting point, we determined our network configuration. It includes number of layers, different layer types and their configurations. At the beginning, we started with a network consisting of a serious of convolutional layers with ReLU activations, pooling and fully-connected layers and do not use any additional techniques like regularization or learning rate scheduling. We used SGD and mean squared error. With this network we examined a serious overfitting as it can be seen from Fig. 1 where train mean absolute error decreased almost down to 0. To prevent this problem, we tried different regularization techniques like dropout and L2 regularization. However, none of them improved the results. We observed that our model cannot capture proper feature representations from the images. Therefore, we use first 5 layers of a pretrained VGG-16 network as a feature extractor [2]. We observed an improvement in the test results. However, even with different learning rates, batch sizes and initialization techniques, loss value was really high and does not decrease well.

Also, we compared 1x1 convolution layers with linear layers. With 1x1 convolution layers, we got similar results but decreased the number of parameters in the network. Therefore, we decided to replace linear layer(s) at the end of the network with 1x1 convolution layers.

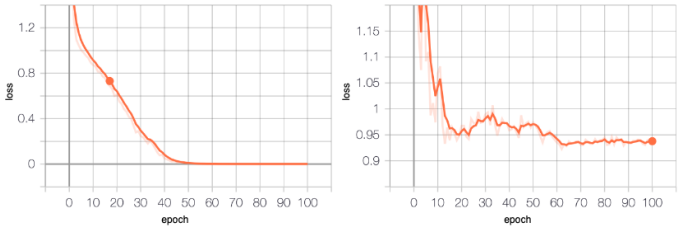

Fig. 1. **Left:** It is train mean absolute error through epochs. It decreases almost down to 0. **Right:** It is validation mean absolute error through epochs. Effect of overfitting to training data can be seen from this plot.

As a result of these experiments, we decided about our network architecture and layer configurations. You can see the final network architecture from Fig. 2.

C. Using Batch Normalization Layers

After we designed the network, we added batch normalization layers after all of the convolution layers except the final one. Batch normalization layers have several advantages like decreasing the internal covariate shift [3] and allowing more efficient and smooth training process [4]. Indeed, when we used it, we observed a better loss value at the beginning and it yielded 0.844 validation error. Therefore, we decided to use Batch Normalization layers during the other experiments. The effect of batch normalization to the loss function can be observed from Fig. 3.

D. Using L1, L2 and Dropout Regularizers

Our model was suffering from overfitting and to prevent it, we tried regularization techniques such as L1 regularization, L2 regularization and dropout with different parameters. We observed L1 and L2 regularizations did not improve the validation error. They did not hurt the model performance as well, therefore we decided to keep L1 regularization with factor of 0.1. Their comparison can be found from Table I. Dropout also helped to reduce overfitting, hence, we deployed it after convolution layers except the last one. According to the conducted experiments, we chose drop rate as 0.1 (Table II). Up to this point number of epochs was 10, however, we changed it to 20 to reduce the additional bias introduced by dropout.

E. Loss Function Tuning

Up to this point, we performed all experiments with mean squared error loss. In that section, we tried mean absolute error and root mean squared error loss functions. According to our experiments mean squared error and root mean squared error loss functions yielded best results (Table III). We further refined the loss function; however, we will discuss them later.

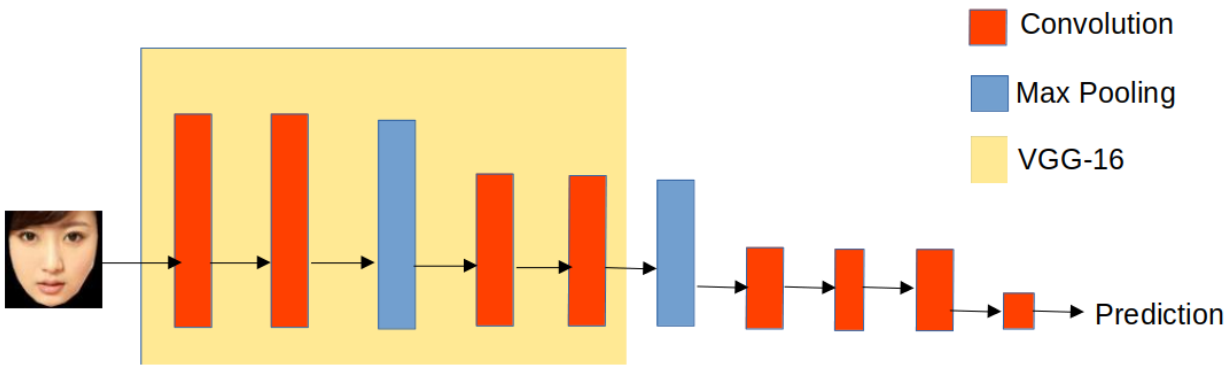

Fig. 2. Our network. We used pretrained VGG-16 layers to extract the features. Every convolutional block that we inserted is followed by dropout and batch normalization blocks, except the last one. We use ReLU as activation function. The network is fully convolutional because we used 1x1 convolution instead of fully connected layers.

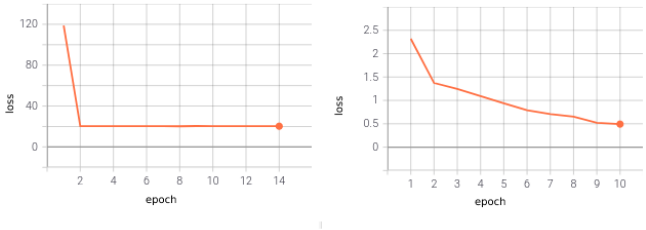

Fig. 3. **Left:** Loss function is not decreasing and it is stuck at 20. **Right:** When we deploy Batch Normalization, we are able to train our network, because the network gets less sensitive the initial values.

TABLE I
COMPARISON OF L1 AND L2 REGULARIZERS WITH DIFFERENT WEIGHT TERMS

Regularizer	Validation MAE
No regularizer	0.844
L1 (0.01)	0.844
L1 (0.03)	0.844
L1 (0.1)	**0.844**
L1 (0.3)	0.856
L2 (0.01)	0.844
L2 (0.03)	0.844
L2 (0.1)	0.844
L2 (0.3)	0.856

F. Weight Initialization

Weight initialization is important because it basically determines where the network starts to the training in the loss space. If it is a bad starting point, then it badly affects the training process. Therefore, as an alternative to Xavier weight initialization, which is what we were trying up to this point, we also tried Gaussian initialization. We got 0.814 mean absolute error on validation set and 0.808 mean absolute error on test set with Xavier initialization. With Gaussian initialization, we got 0.875 mean absolute error on validation set and 0.825 mean absolute error on test set. Therefore, we decided to continue with Xavier initialization.

TABLE II
COMPARSION OF DROP RATES

Drop Rate	Validation MAE
No Dropout	0.844
0.1	**0.814**
0.15	0.860
0.05	0.844
0.3	0.856

TABLE III
COMPARISON BETWEEN DIFFERENT LOSS FUNCTIONS

Loss Function	Validation MAE	Test MAE
Mean Absolute Error	0.837	0.812
Mean Squared Error	0.814	0.808
Root Mean Squared Error	**0.811**	0.815

G. SGD vs Adam Optimizer

Up to this point, we were using SGD optimizer with momentum 0.9 and learning rate 0.001. We also tried Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$ as it is suggested in the original paper [1], however, we did not observe any improvement. For both optimizers, we experimented with higher learning rates. The summary of the experiments is in Table IV.

Because SGD performed better on validation set, we decided to keep using it. Furthermore, although higher learning rate did not reduce validation error, it decreased test error (Table V).

H. Regression Layer

In the final layer of the network, we used ReLU activation function as we knew that given attractiveness labels are greater than zero. To better understand if this was the correct approach, we removed the ReLU activation at the output layer. We observed that removal resulted in 0.828 validation error, whereas the previous setting was resulting in 0.814 validation error (Table VI). Therefore, at the rest of the experiments we kept the last ReLU layer.

TABLE IV
COMPARISON BETWEEN SGD AND ADAM OPTIMIZERS.

Optimizer	Validation MAE	Test MAE
SGD	**0.814**	0.808
Adam	0.816	0.780

TABLE V
COMPARISON BETWEEN DIFFERENT LEARNING RATES FOR SGD OPTIMIZER.

Learning Rate	Validation MAE	Test MAE
0.001	0.814	0.80
0.003	**0.814**	**0.76**

I. Batch Size

We were using 32 as batch size simply because this was the highest number that can fit to the GPU memory. We experimented with different settings and found that validation error increases when we decrease batch size (Table VII).

J. Further Refinements to Loss Function

Because we found that both mean square error and root mean square error were beneficial, we decided to explore the weighted average of them. According the our experiments (Table VIII) , multiplying mean square error with 0.75 and root mean square error with 0.25 yielded to the best result. We can define the difference between MSE and RMSE as follows: MSE penalizes larger errors harder than RMSE. Therefore, taking weighted sum of these two loss functions allows us to find an optimal penalization between them.

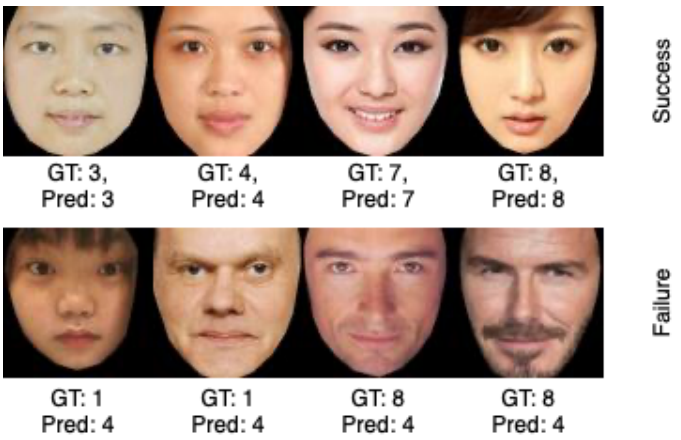

Fig. 4. Success and failure cases with their ground truth and predicted attractiveness values.

III. FINAL RESULTS

When we analyze the final results, the highest MAE values are seen for 1 and 8 (Table IX). Indeed, when we look at images, their ground truth labels and predicted labels, some large error margins can be seen for 1 and 8. The failure cases that can be seen from Fig 4 are the ones which have the largest error margins among them. Data distribution in the training dataset may be the reason for this problem. When the Table X is analyzed, it is seen that 1 and 8 are the labels with the lowest number of samples. Therefore, the model may not see many images with label 1 and 8 and cannot learn a pattern for these labels sufficiently. A possible remedy for this problem is increasing the number of samples for these labels and allow the model to see more images with these labels.

TABLE VI
COMPARISON BETWEEN THE NETWORKS WHEN USING AND NOT USING RELU ACTIVATION FOR OUTPUT LAYER.

Network	Validation MAE	Test MAE
With Final ReLU	**0.814**	0.808
Without Final ReLU	0.929	0.873

TABLE VII
VALIDATION SCORES FOR DIFFERENT BATCH SIZES.

Batch Size	Validation MAE	Test MAE
8	0.838	0.817
16	0.834	0.816
32	**0.814**	0.761

TABLE VIII
COMPARISON BETWEEN DIFFERENT WEIGHT TERMS. IT IS FOR TUNING THE WEIGHTED SUM OF RMSE AND MSE.

MSE Weight	RMSE Weight	Validation MAE	Test MAE
0.50	0.50	0.835	0.793
0.70	0.30	0.811	0.816
0.80	0.20	0.813	0.792
0.75	0.25	**0.804**	0.803

TABLE IX
MAE FOR EACH ATTRACTIVENESS LABEL

Attractiveness Label	MAE
1.0	1.36
2.0	0.81
3.0	0.55
4.0	0.77
5.0	0.71
6.0	0.92
7.0	0.85
8.0	1.04

TABLE X
NUMBER OF SAMPLES FOR EACH ATTRACTIVENESS LABEL IN THE TRAINING DATASET

Attractiveness Label	Number of Samples
1.0	240
2.0	576
3.0	903
4.0	596
5.0	312
6.0	336
7.0	376
8.0	216

IV. Conclusion

As a conclusion, we have performed many experiments to compare different techniques with each other with different parameters. Using a pretrained VGG-16 network and batch normalization layers can be seen as a boost for our experiments. They improved the results and made the training process more efficient. Also, adding dropout layers after convolution layers decreased the effect of overfitting and allowed us to get better validation and test scores. Since it is a regression task and labels are greater than 0, we tried to use ReLU activation at the output layer and it yielded better results. Finally, weighted sum of MSE and RMSE gave the best results after we selected the weight terms as a result of several trials. Our best model gives 0.804 MAE for the validation set and 0.803 MAE for the test set.

References

[1] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015.

[2] Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2015.

[3] Christian Szegedy Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] Andrew Ilyas Shibani Santurkar, Dimitris Tsipras. How does batch normalization help optimization? *arXiv preprint arXiv:1805.11604*, 2019.