CENG 483

Introduction to Computer Vision
Fall 2023-2024

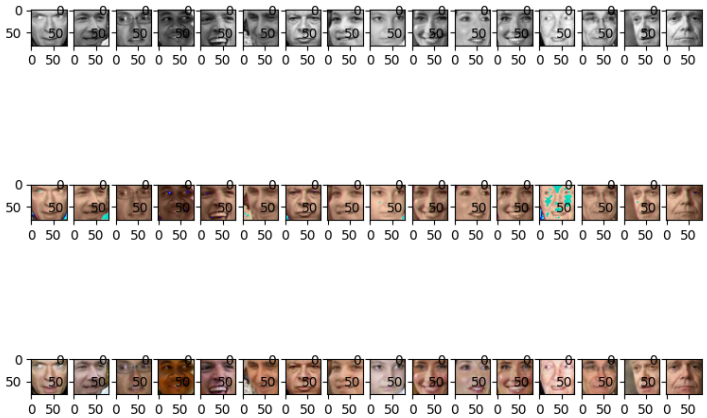
Take Home Exam 3: Image Colorization

Full Name: Furkan Numanoğlu
Student ID: 2448710

Please fill in the sections below only with the requested information. If you have additional things to mention, you can use the last section. Please note that all of the results in this report should be given for the **validation set** by default, unless otherwise specified. Also, when you are expected to comment on the effect of a parameter, please make sure to **fix** other parameters. You may support your comments with visuals (e.g., loss plot).

0.1 My Results

(a) Layer:1 Kernel:2



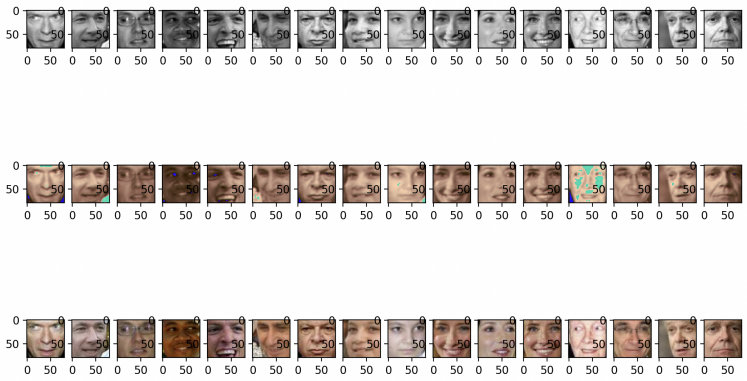
(b) Layer:1 Kernel:4

(c) Layer:1 Kernel:8

Figure 1: Number Of Convolutional Layer:1

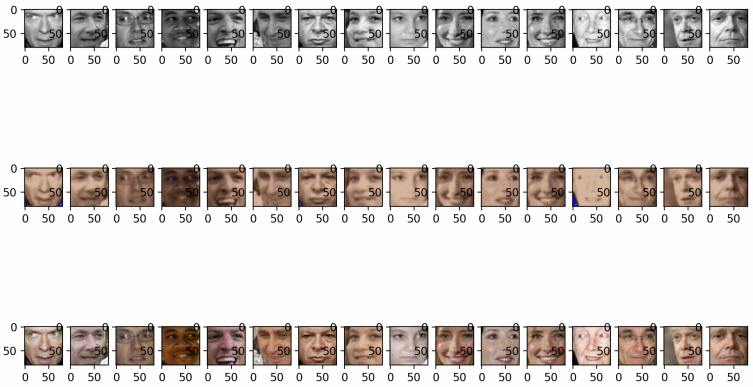
(a) Layer:2 Kernel:2

(b) Layer:2 Kernel:4



(c) Layer:2 Kernel:8

Figure 2: Number Of Convolutional Layer:2



(a) Layer:4 Kernel:2

(b) Layer:4 Kernel:4

(c) Layer:4 Kernel:8

Figure 3: Number Of Convolutional Layer:1

1 Baseline Architecture (30 pts)

Based on your qualitative results (do not forget to give them),

- Discuss effect of the number of conv layers:

  A smaller number of convolutional layers can reduce the number of parameters of the model and a simpler model can be obtained. In this way, faster results can be obtained. However, as the complexity of the model decreases, its ability will generally decrease relative to more complex structures. Reducing the complexity of the model increases its applicability due to issues such as time and resources, but a model with low ability and far from the desired result will be meaningless. However, when we keep the number of kernels constant at 4 and focus on the results of changes in the number of layers, we see the following:
  While the number of layers is 4, the model is much more blurred than the original photo and is not at a better point in terms of coloring. When the number of layers is 1, meaningless colored pixels stand out. I think the reason for this is that since my model was prepared with a simple approach, as the number of layers increases, we need to try different methods and be able to handle this complexity. By these different methods I mean: pooling, different activation functions, etc. However, these were not done in the homework.
  As a result, I think that the optimum result in terms of training time and talent is obtained with 2 layers.
  Additionally, I shared the qualitative results I used in making this evaluation at the top of the assignment.

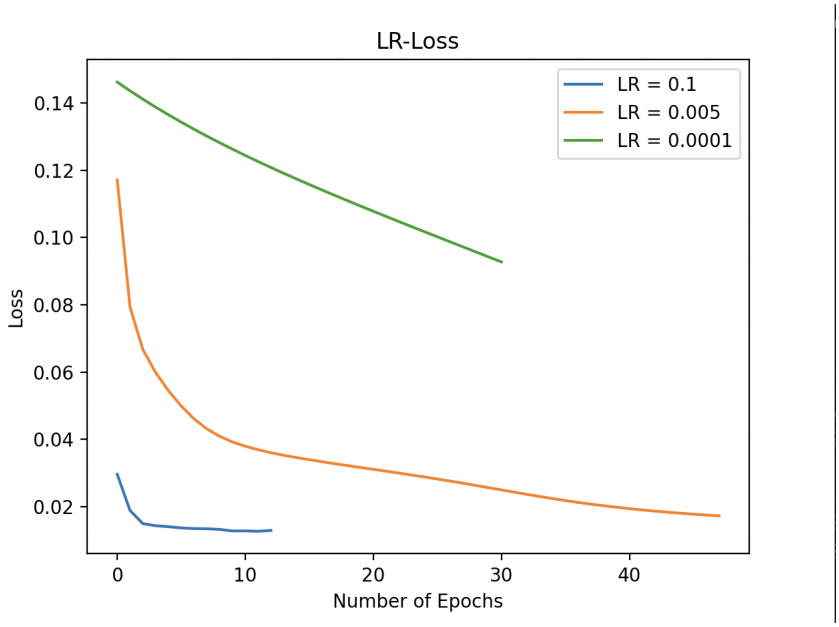
- Discuss effect of the number of kernels(except the last conv layer):
  Fewer kernels can lead to lower learning capacity and a less complex model. It may be effective when learning simpler patterns, but will be poor at perceiving complex patterns. This is due to the number of features.

  Increasing the number of kernels can ensure that the model has a higher learning capacity. However, the possibility of overfitting increases even more, so it is important to use a good trimming strategy. Based on my observations and the results I obtained, I noticed this: as the number of kernels increases, there is some improvement in the details in the final photos of the model. What I mean by details is the reduction of blurriness and a slight improvement in the realism of the colors, albeit at a barely noticeable level. However, due to increasing complexity, the amount of unexpected corruptions may occur as the number of kernels increases. This is not interesting to come by because as complexity increases, it can get out of control. However, in general, except for minor unexpected disruptions, I can say that we reached more neurons and obtained better results with the increasing number of kernels. Although the visible result is better, after a point it becomes meaningless to increase the number of kernels when considering issues such as time and resource usage.

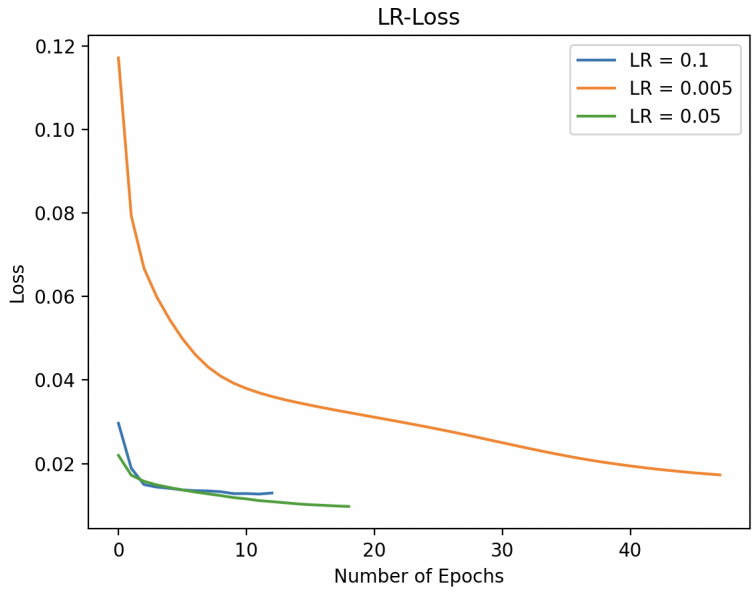
- Discuss effect of the learning rate by choosing three values: a very large one, a very small one and a value of your choice:

  In Conv layer:2, Kernel number:4 configuration,
  When 0.0001 is selected as the learning rate, the epoch remains at 78 and the loss remains at 0.02. When the learning rate is selected as 0.1, the epoch remains at 12 and the loss remains at 0.01. When the learning rate is selected in the middle, it either gives a performance as good as 0.1 or worse. At this point, a situation occurred that contradicted my theoretical knowledge. At first glance, this is contrary to the high - average - low learning rate curve that I see in the sources

and which also appears in the exam. I should not choose the learning rate too low for this model, since 0.1 is the one that works best within the learning rate range allowed to us. However, another situation I encountered in my observations is this: As the learning rate increases, occasional jumps in loss can be observed.
When the learning rate is chosen low, it can be generalized that the model learns more slowly but steadily and accurately, but if the learning rate is chosen too low, the model may get stuck and may give results that are far from the desired result at the desired time due to the long working time. Therefore, the logic of being stable but slow is not correct.



As you can see, 0.1 gives good performance. However, due to the peak points in my few attempts, I will choose a slightly higher version for safety reasons.

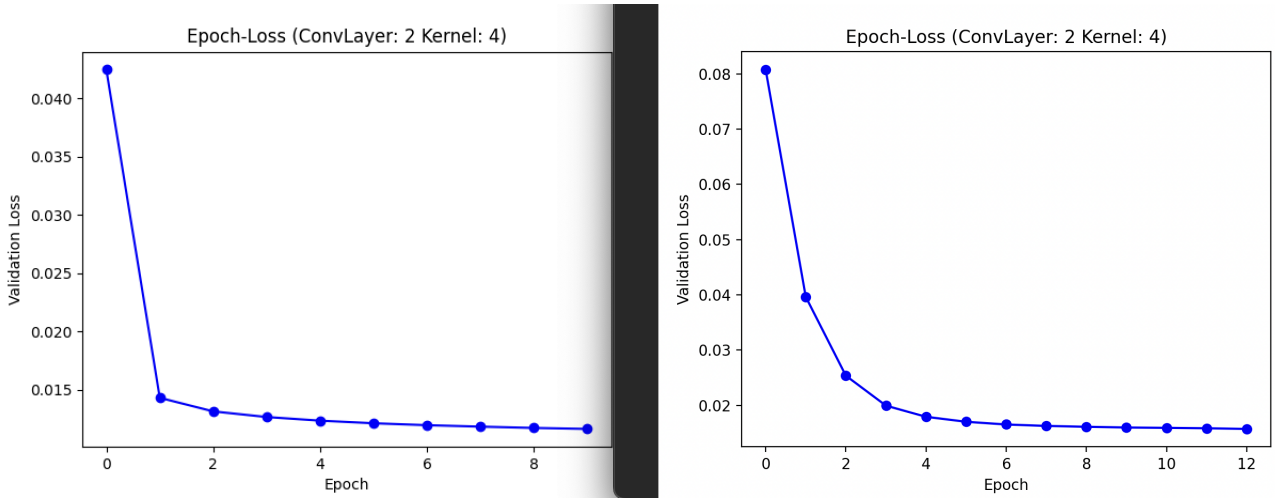


As can be seen, 0.05 is a reliable, fast and successful learning rate value.

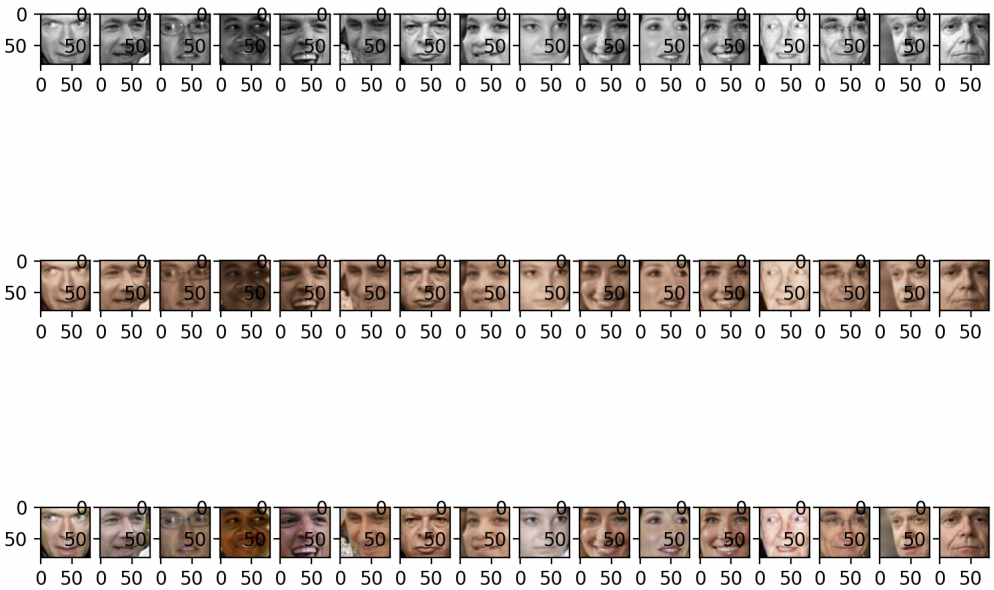
2 Further Experiments (20 pts)

Based on your qualitative results (do not forget to give them),

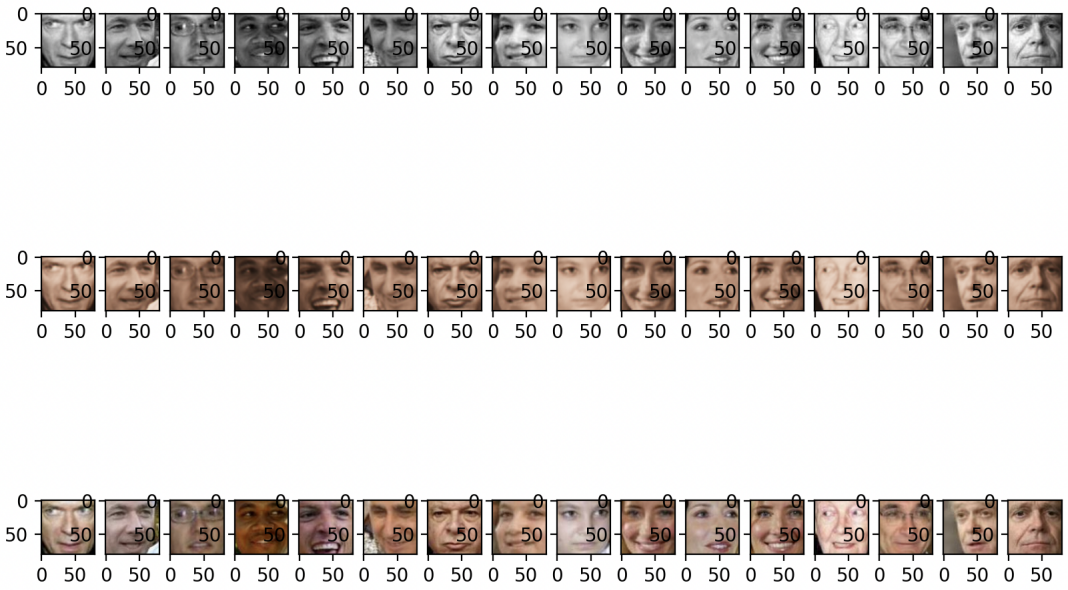
- Try adding a batch-norm layer (torch.nn.BatchNorm2d) into each convolutional layer. How does it affect the results, and, why? Keep it if it is beneficial.



The batch on the left is not normalized, the one on the right is batchnormalized. Both are with tanh applied.
I haven't had a better result. The loss function both fell more slowly and was higher at its lowest point than if it were not batchnormed. And I couldn't see any visible improvement in the pictures.



Batch Normalization + TanH



Just TanH
My guesses about the possible reasons for this:

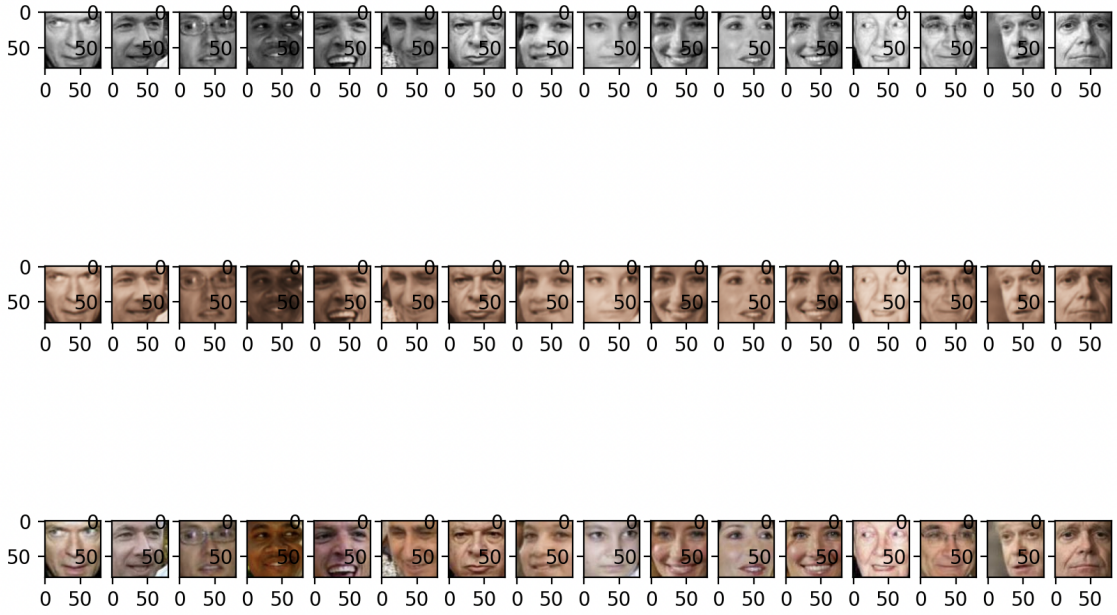
It may be less because the minibatch size is 16. The fact that the normalized batch contains few elements may have affected its behavior.
With batch normalization, the initial values of the weights may become important. It is important to use a good starting strategy to start the weights properly. However, I do not use a special method to initialize weight, Python handles it itself. A different configuration can be tried.

Batch normalization can help train the network faster. Because normalization is performed for each mini-batch, it can contribute to a more stable update of the gradients. I also observe that on my model.

In addition, I found that batch normalization works better when the learning rate is low. Since very high values were reduced thanks to the normalization process and all values were confined to a certain range, although the learning rate was lower, it worked both faster and more stable than the non-normalized version. I have seen that Batch Normalization is seriously effective when the learning rate is chosen relatively low.

- Try adding a tanh activation function after the very last convolutional layer. How does it affect the results, and, why? Keep it if it is beneficial.

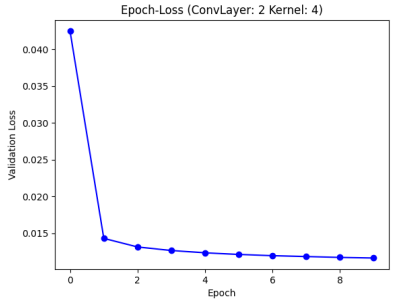
In terms of destination, not much has changed numerically. The loss function graph is almost the same. However, there is a decrease in strange colorations that we can easily perceive.
With Tanh:

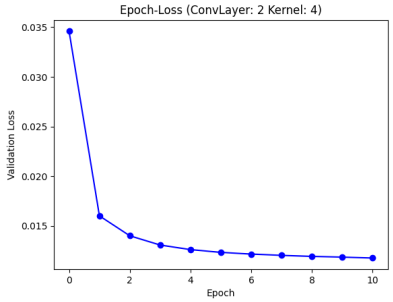


Without Tanh:

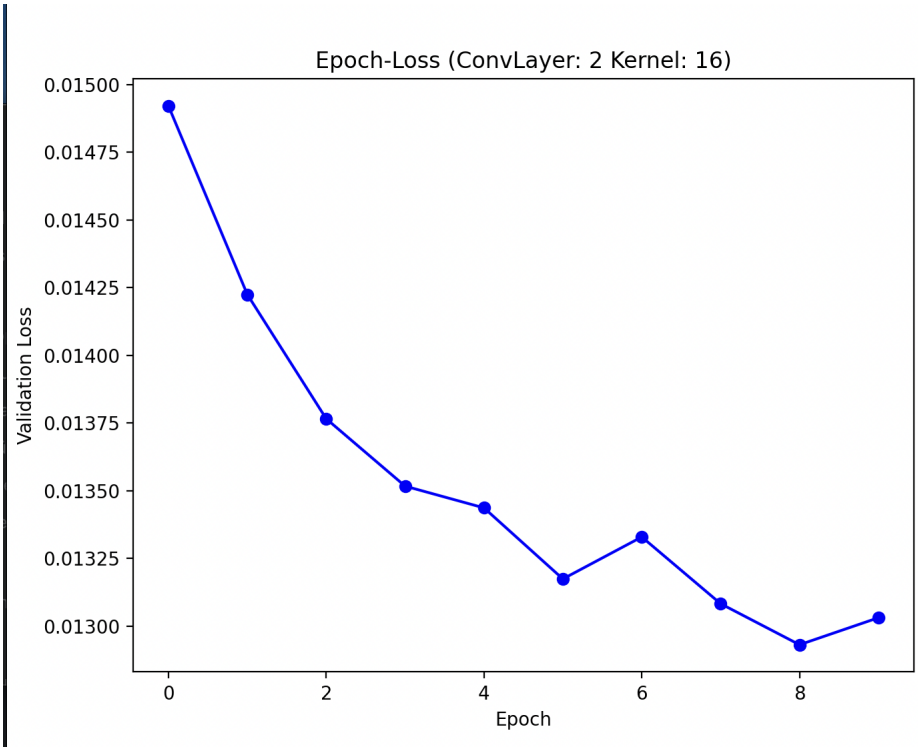
As can be seen, more consistent and realistic colors were obtained thanks to TanH. The reason for this is that we display our normalized input images in a normalized way. TanH's zero-centering was beneficial at this point. Since we multiply values smaller than zero by 0 and larger values linearly with Relu, we were able to find values at the 1 border of the range 0 and [-1,1]. In TanH, we obtained zero centered data. I will share the loss function below, but not much change was

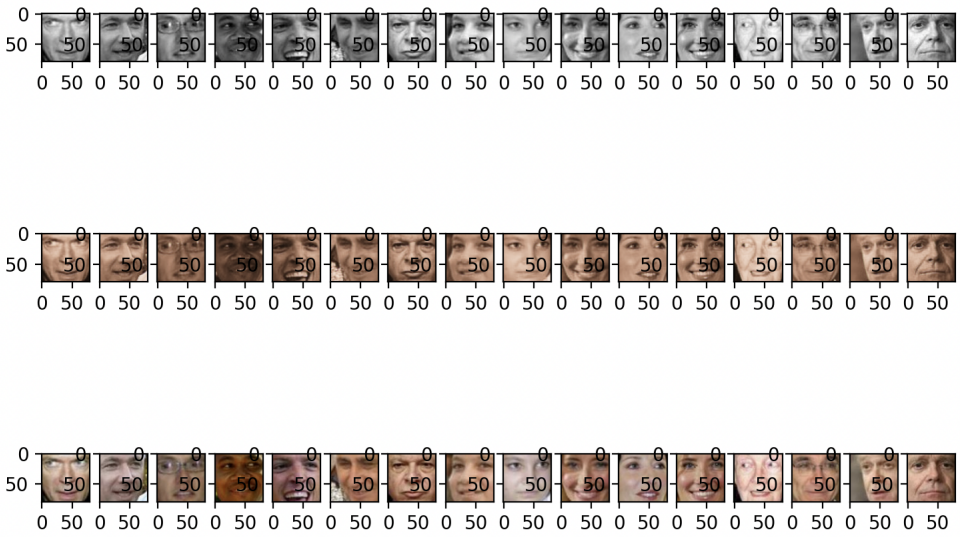
observed.
The tanh on the left has been applied, the tanh on the right has not been applied.





- Try setting the number of channels parameter to 16. How does it affect the results, and, why? Keep it if it is beneficial.





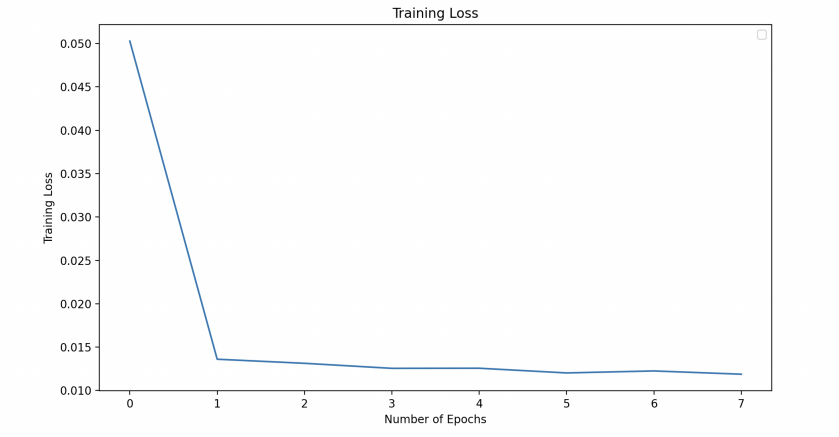
We expected that adding more features would lead to better learning, but this did not happen. We are faced with a situation that is not bad, but we can get better ones with a much smaller number of kernels. In some cases, adding more features doesn't have to work well. A high number of mappings creates a meaningless situation. Unnecessary complexity returns to us negatively.

3 Your Best Configuration (20 pts)

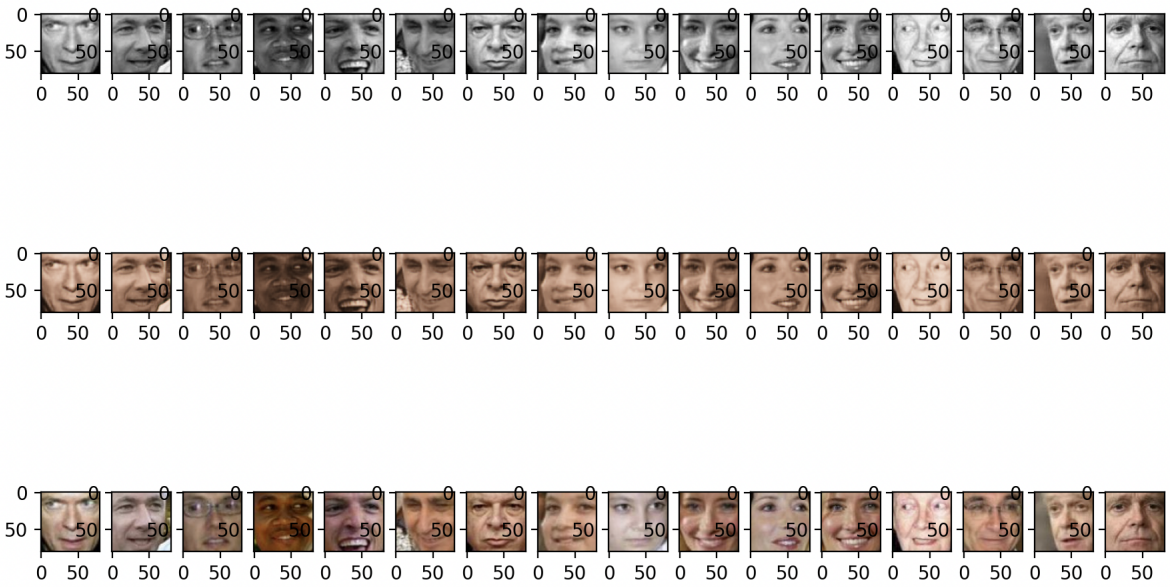
Using the best model that you obtain, report the following:
Learning rate: 0.05
Regularization: 0.001
Batch normalization did not applied
Number of Conv Layer: 2 Number of Kernel: 4

- The automatically chosen number of epochs (what was your strategy?):
  Automatically chosen epoch: 9
  My strategy is to terminate the training if I get a validation loss that is very close to or greater than the previous 5 epochs in a row. What I mean very closely is that I handle the incoming validation loss number as 2 significant digits. I make all my comparisons as 2 significant digits. Therefore, when minimal differences occur, I ignore them.

- The plot of the training mean-squared error loss over epochs:



- The plot of the validation 12-margin error over epochs (see the3 text for details):
- At least 5 qualitative results on the validation set, showing the prediction and the target colored image:



- Discuss the advantages and disadvantages of the model, based on your qualitative results, and, briefly discuss potential ways to improve the model:
  The problem with my model is that it cannot go further than one point. I have to try different ways to overcome this (I mentioned below). When the model kernel is added, it reaches saturation and produces unexpected colors. A more detailed architecture is required to solve this.
  Weight initialization can be done better (for example, Xavier can be tried). Regularization is not discussed much in the homework, but it can be tried.
  However, I think it will be in the real improvement model architecture. Different activation functions, cross-validation etc. I'm not sure if adding more kernels and layers would be very helpful

at this point because I couldn't get any effective improvement when I tried higher kernels and layers. Previously trained models can be taken and placed on top. Pooling can also be tried for improvement.

4 Your Results on the Test Set (30 pts)

This part will be obtained by us using the estimations you will provide. Please tell us how should we run your code in case of a problem: You can look at the readme file.

5 Additional Comments and References

(if there any)