CENG 483

Introduction to Computer Vision
Fall 2023-2024
Take Home Exam 3: Image Colorization

Full Name: AHMET EREN KARDAŞ
Student ID: 2448579

1 Baseline Architecture (30 pts)

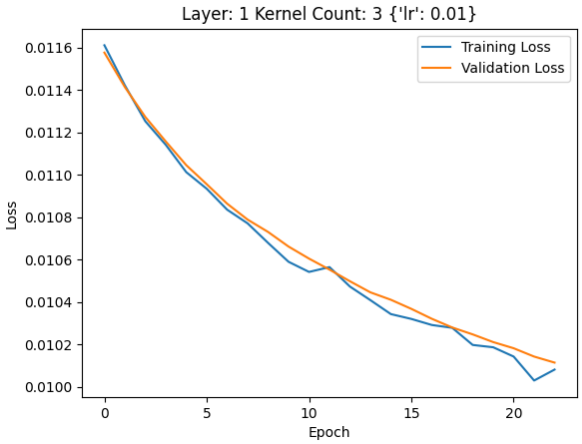
In this section I have tried many configurations by using grid search on hyper-parameters which are given below
kernel size: 2, 4, 8
layer count: 1, 2, 4
learning rate: 0.1, 0.01, 0.001, 0.0001.

After the results I have took some of them the compare and share the results.
Additionally, as I am color-blind I am not able to understand the slight changes in the image results, so I will make my derivations based on the plot mostly if there is not dramatic difference between the results of configurations on images. Also, to share the plot's main curve more clear, I have removed the first losses of both training and validation, this does not effect the results, but make our plots more clear to make conclusion.

1.1 Effect of the Number of Conv Layers

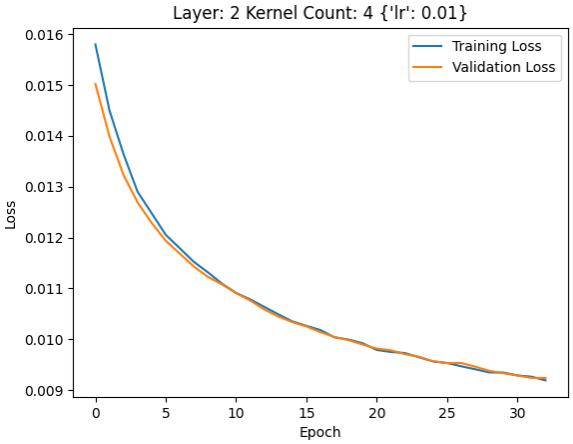
Firstly, from the results of the experiments I conducted, we cannot conclude any direct and strict cause-and-effect relationship regarding the number of layers applied to any configuration. However, we can still say that depending on the other parameters, we can observe a some effect of the number of layers differently in various configurations.

First case is that when we keep number of kernel and learning rate as 4 and 0.01 consecutively, we can see changes in our losses between the configurations respect to layer count as below.



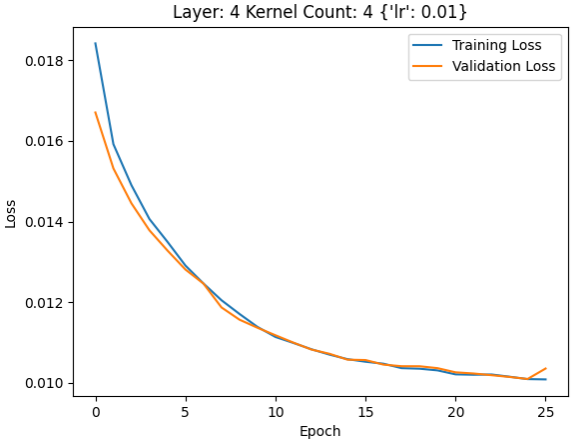
1 Layer

1 Layer



2 Layer

2 Layer

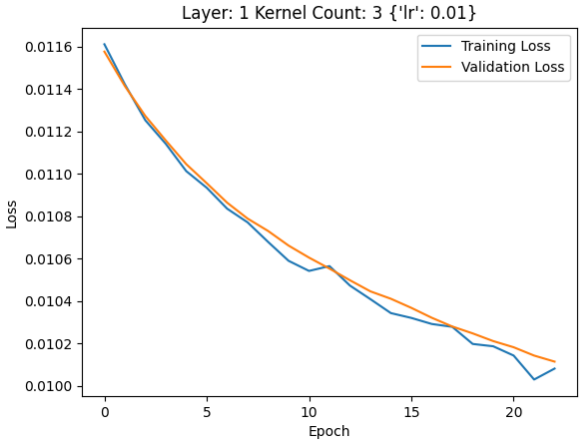


4 Layer

4 Layer

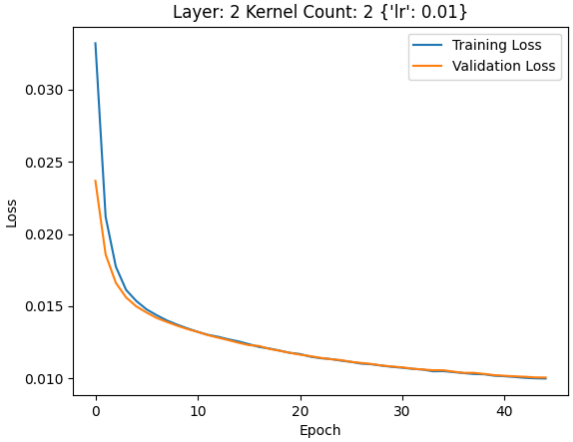
From the figures given above, it can be observed that increasing the number of layers may contribute to smoothing the loss curve. This could be performance enhancing when we merge with other parameter adjustments. However, the effect is not clear enough, and drawing a definitive conclusion is not convenience.

In the second scenario, when maintaining the number of kernels and learning rate at 2 and 0.01, respectively, variations in losses between different layer counts can be seen below.



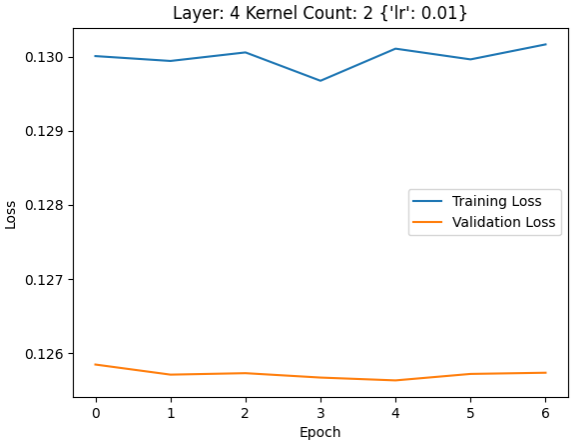
1 Layer

1 Layer

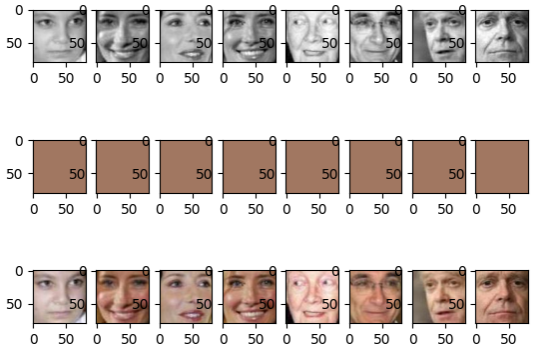


2 Layer

2 Layer



4 Layer



4 Layer

In the provided configuration space above, the unexpected behavior of the 4-layer setup is can be visible. This change in layer count negatively impacted both our loss and the model. The reason could be that our network or our objective is not well-suited for a 4-layered and 2-kernel configuration.

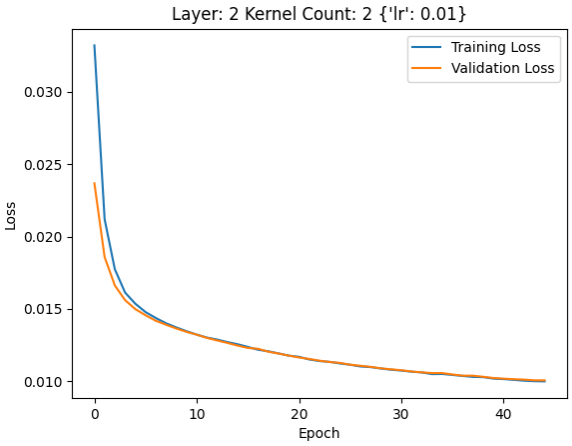
To summarize, we cannot directly state that increasing the layer count increases or decreases DNN performance. As seen in Experiment 2, the 4-layered version has a much worse effect than the 2-layered

version. Therefore, we can conclude that an increase in layer count has different effects on different configurations.

1.2 Effect of the Number of Kernels

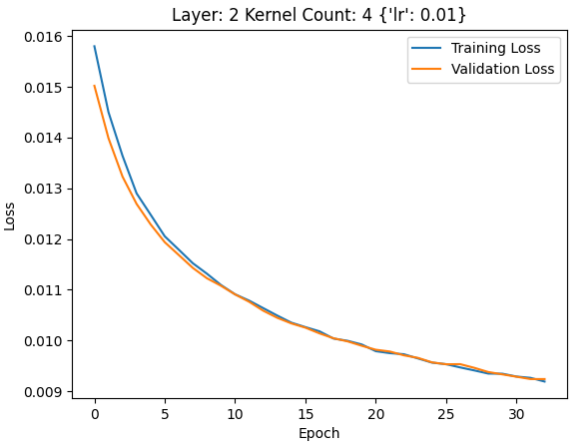
From the results of the experiments I conducted, we can conclude that an increase in kernel count leads to a better learning curve in some configurations. When we adjust the other hyper-parameters accordingly, configurations with a high number of kernels give better results.

First case is that when we keep number of layer and learning rate as 2 and 0.01 consecutively, we can see changes in our loss' curves between the configurations respect to kernel count as below.



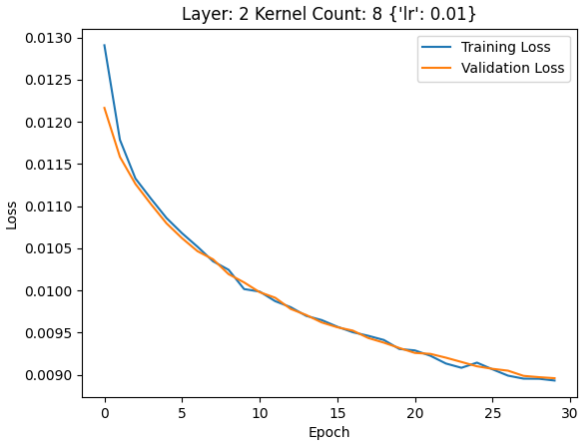
2 Kernel

2 Kernel



4 Kernel

4 Kernel

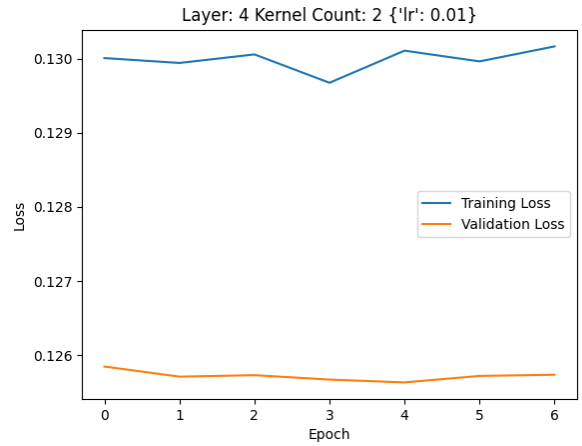


8 Kernel

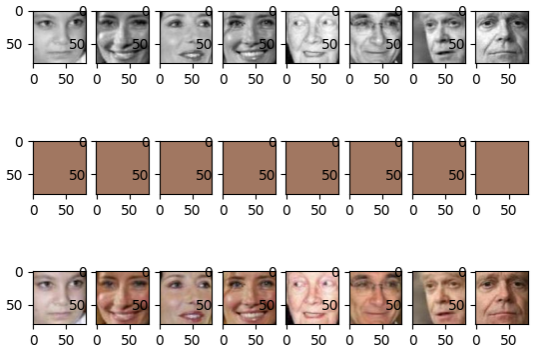
8 Kernel

In the provided configuration space above, it's evident that an increase in kernel count leads to an improvement in the learning curve. As observed in the 2-Kernel setup, the learning rate tends to decrease faster than in configurations with higher kernel counts. With each increment, there is a noticeable evolution of the learning curve in a good manner. This presents a promising model that we can further optimize.

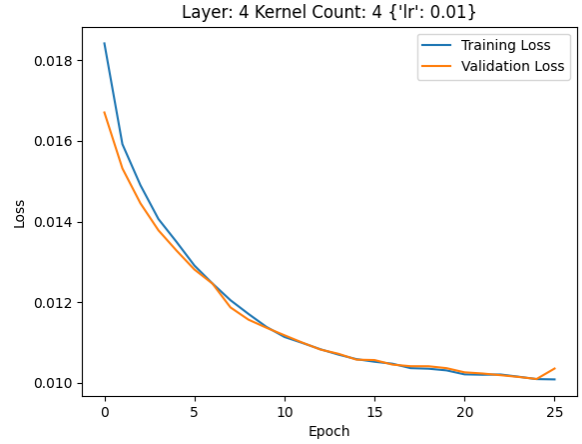
Second case is that when we keep number of layer and learning rate as 4 and 0.01 consecutively, we can see changes in our loss' curves between the configurations respect to kernel count as below.



2 Kernel

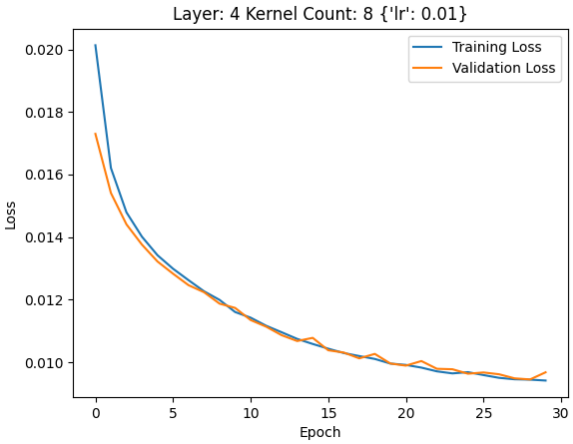


2 Kernel



4 Kernel

4 Kernel



8 Kernel

8 Kernel

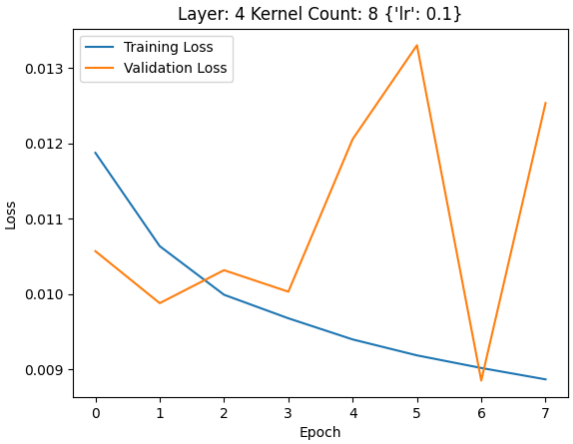
The experiment results provided above also support our conclusion. A dramatic difference between kernel counts 2 and 4 is easily noticeable, with the increase in kernel count significantly improving the curve at the first change. However, it's important to note that there is not a substantial change in the loss curve with the transition from kernel count 4 to kernel count 8.

In summary, the kernel count should be arranged according to the some threshold observed in the experiments. As we saw in the results, a kernel count of 2 is a very inconvenience choice, while there are only slight differences between counts of 4 and 8. Taking both experiments into consideration, it can be concluded that a kernel count of 8 is more promising and appears to be more optimizable.

1.3 Effect of the Learning Rate

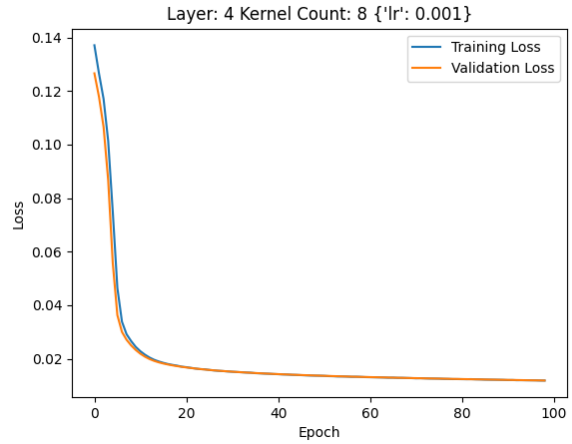
From the experiments I have conducted, it can be observed that, as expected, higher learning rates result in a dramatic decrease at the beginning and no further learning. On the other hand, lower learning rates yield linear curves that exhibit continuous learning. However, this can lead to either very slow learning that does not progress significantly or a learning curve that becomes stuck at some point, which is undesirable.

First case is that when we keep number of layer and kernel count as 4 and 8 respectively, we can see changes in our loss' curves between the configurations respect to learning rate as below.



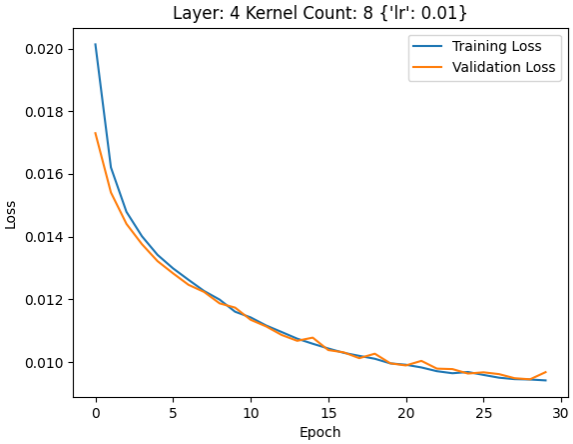
0.1 Learning Rate

0.1 Learning Rate



0.001 Learning Rate

0.001 Learning Rate

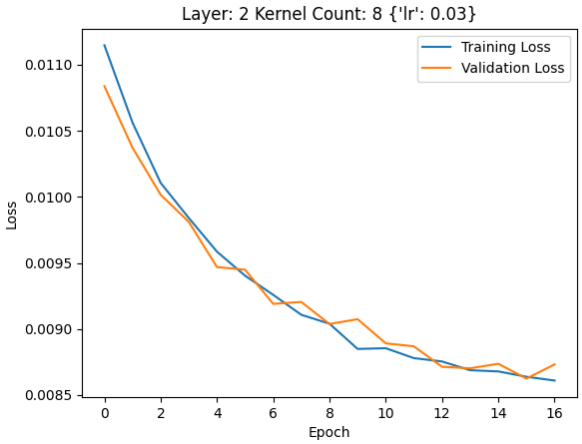


0.01 Learning Rate

0.01 Learning Rate

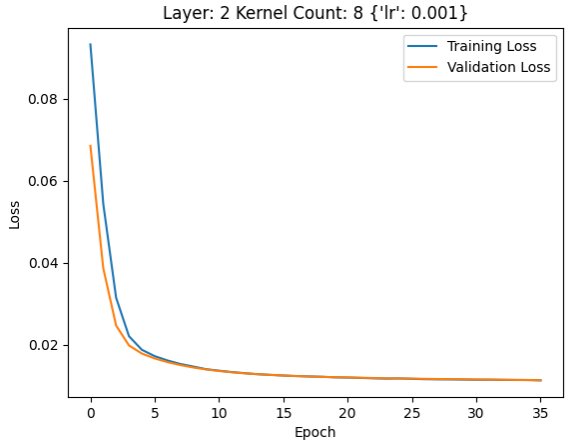
From the experiment results given above, we can conclude what we discuss in the start. If the learning rate is too high then we can easily see significant drop in the loss, but also some abnormal curves that could not continue learning. The second case is very low learning rate, and this leads good learning at the beginning but it stucks at some point and can not continue to learn. These two curves are exactly like what we expected before the experiment. So, we can try some learning rate value between these to consider if we can derive good learning curve, and I select middle of them 0.01 as learning rate and we can easily see that it is more smooth learning curve and more optimizable respect to others.

Second case is that when we keep number of layer and kernel count as 2 and 8 respectively, we can see changes in our loss' curves between the configurations respect to learning rate as below.



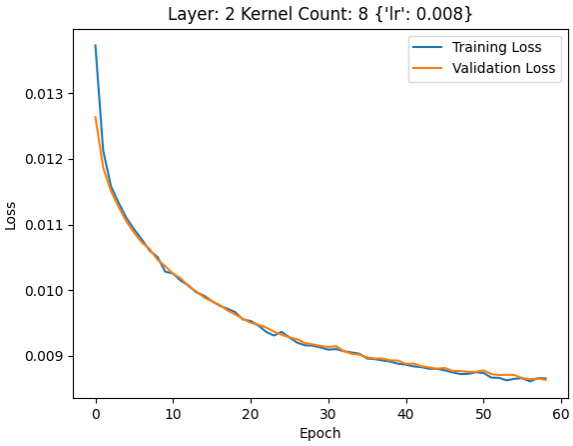
0.03 Learning Rate

0.03 Learning Rate



0.001 Learning Rate

0.001 Learning Rate



0.008 Learning Rate

0.008 Learning Rate

Experiment results above also justify our claim at the beginning. In this experiment I select as 0.008 as good learning rate because my aim was find some smooth curve between 0.03 and 0.001 results, and 0.008 fits perfect in this case

2 Further Experiments (20 pts)

After the experiments I have done on my Baseline Architecture, I have found two promising configurations to continue with which are :

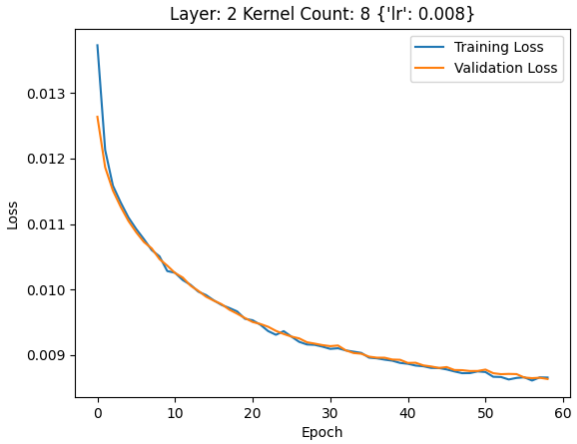
First,
Layer count: 2
Kernel count: 8
Learning rate: 0.008

Second,
Layer count: 4
Kernel count: 8
Learning rate: 0.008

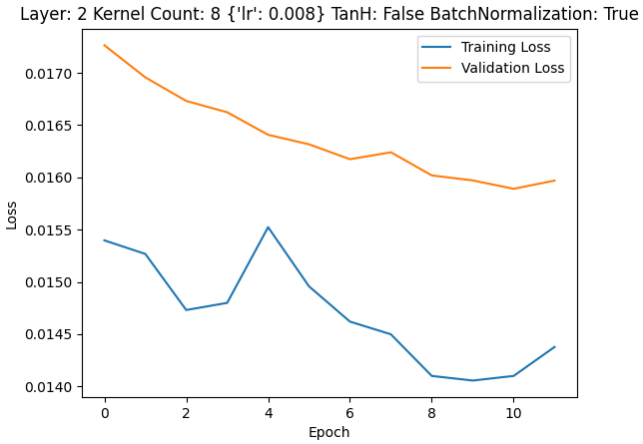
I have derived these values by greedily trying all the configurations in some range, and after all I found 0.01 as around some good points, so I have looked for an another value between 0.01-0.001 ,and 0.008 is good enough to continue.
From these two configurations I decided to continue with the first one because even they are too close to each other, layer count 2 is going lower loss values.

2.1 Adding Batch-norm layer

I have added batch-norm layer to my First configuration above, and results are given below.



Base

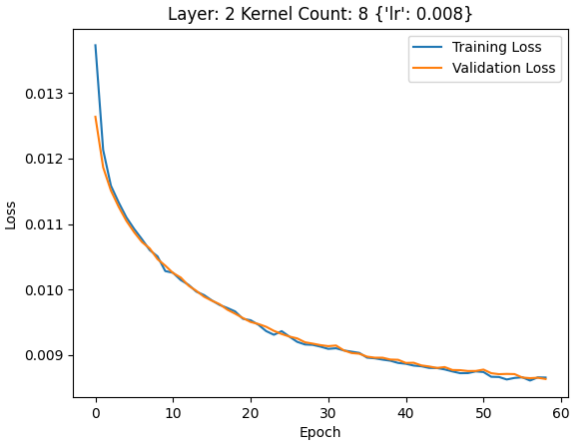


Batch Normalization

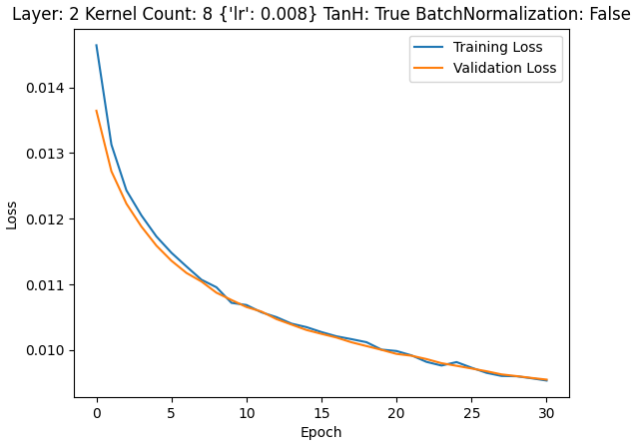
From the results above we can say that batch-normalization is preventing our model learning. These results are unexpected when we consider batch-normalization logic in general, but in our case, we are using 2 layer only and our network is not complicated and well structered. As a result of these reasons, batch-normalization does not work well in our system. I will not continue with batch-normalization as it is effecting our model inconvenience way.

2.2 Adding Tanh Activation Function

I have added Tanh activation function to end of my last layer, and results are given below.



Base

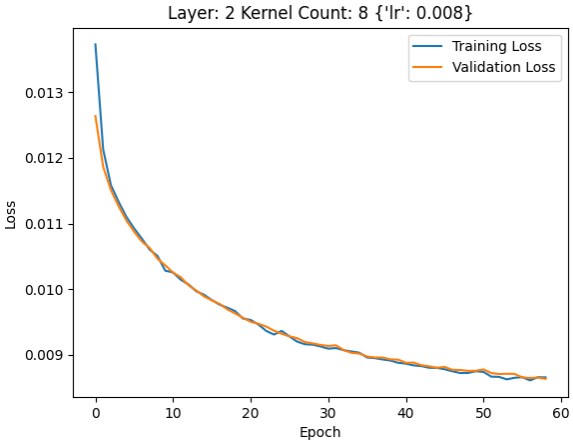


Tanh

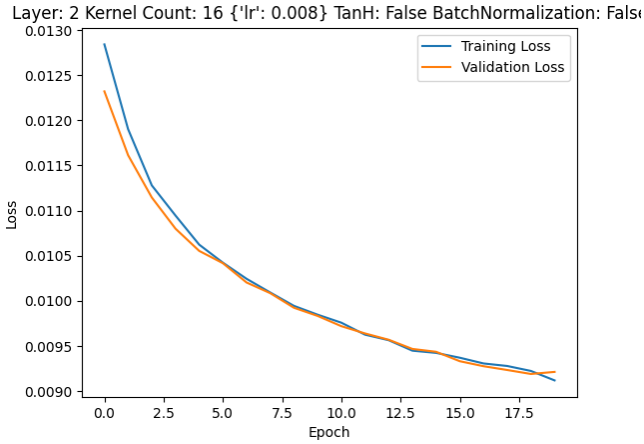
From the results are given above, we can say that adding Tanh activation function at the end is not effecting significantly our Network performance, yet it still smoothing our curve. Normally when we add Tanh, it may effect the system in a bad manner because of it's saturating feature, but as we discussed in batch normalization part, our network is low level structured and not complex, so Tanh effect in this case have some improvements on our system.

2.3 Setting the Number of Channels Parameter to 16

I have set the number of kernel the 16, and results are given below



Base

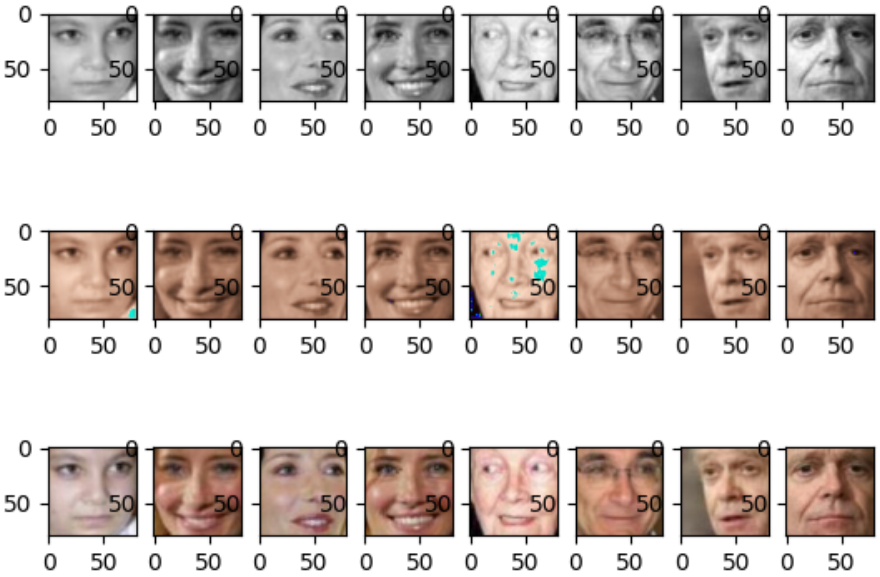


16 Kernel

From the results are given above, we can say that increase in kernel number makes our learning curve more linear, and give some better results in some images. As expected, increasing number of kernel may tends to increase in performance in general because it allows us more parameters to optimize, and this may have a good effect on overfitting. However, in our network there is not such a system to see this improvement clearly. Still it is beneficial for my configurations, and I will use it.

2.4 Conclusion

I have implemented all of the additions we discussed above, and results of them differently given below on validation data set.



Base

Batched Normalization

Tanh Activation

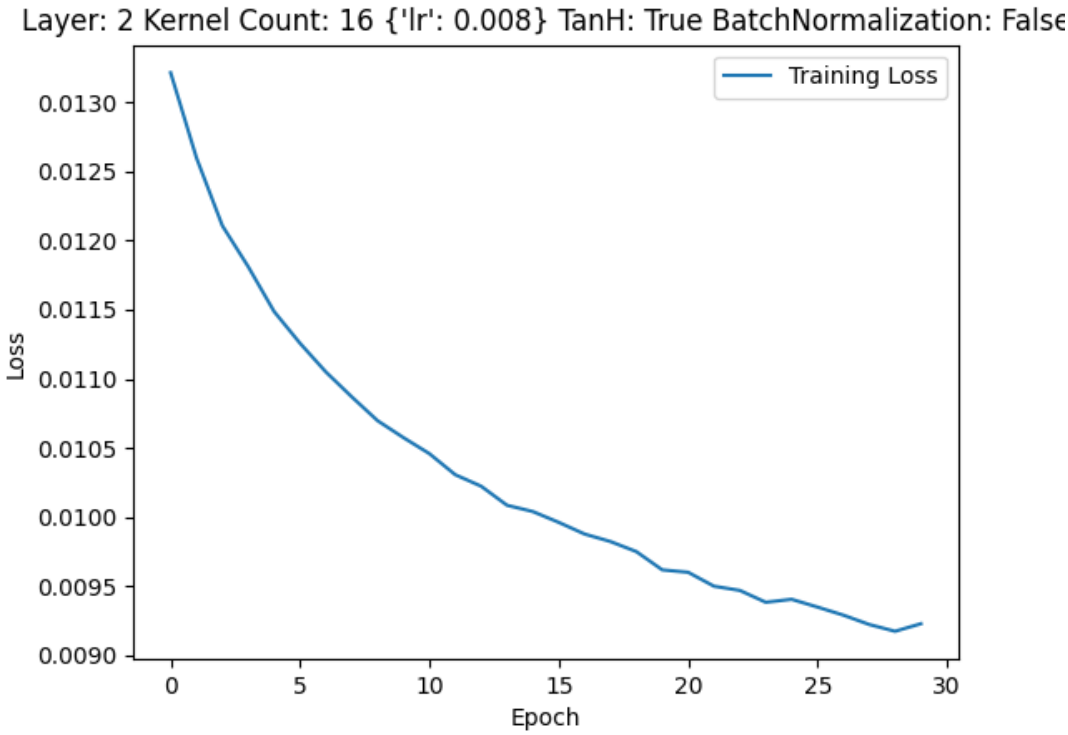
16 Kernel

From the results of different 4 configurations, we can say that results of tanh and 16 kernel are better than the base one, and batched-normalization results with some pixel errors. Therefore I will continue with tanh and 16 kernel.

3 Your Best Configuration (20 pts)

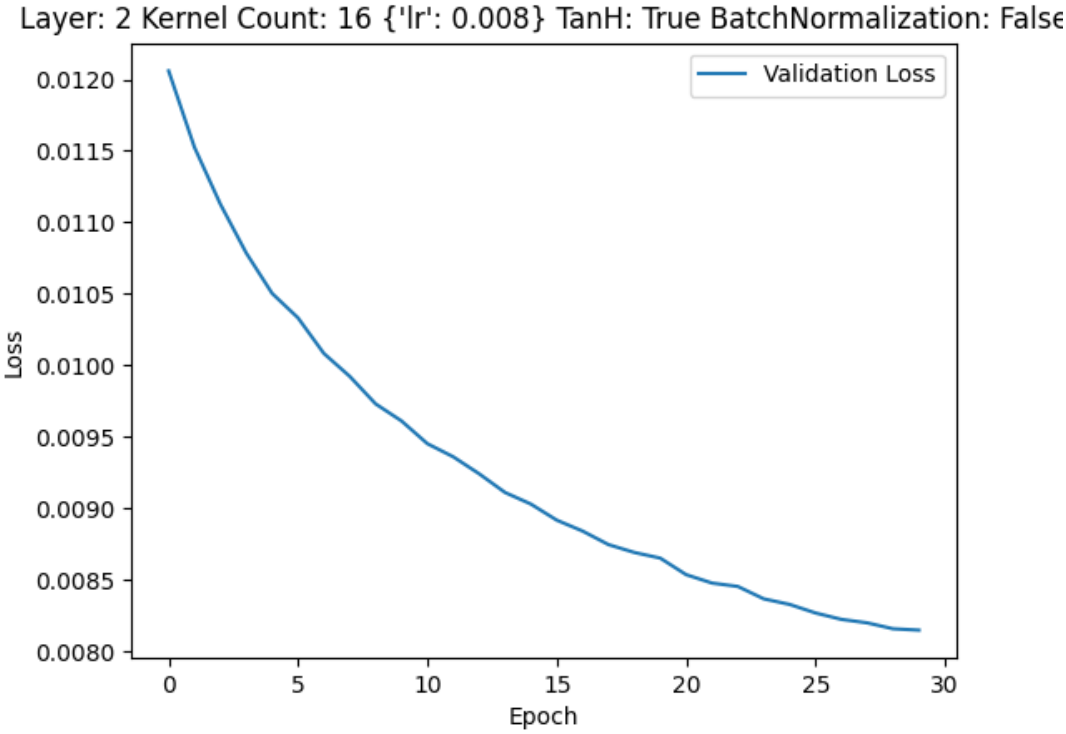
Best model I have obtained is:
Layer count: 2
Kernel count: 16
Learning rate: 0.008
Tanh: True
Batch-normalization: False

- **The automatically chosen number of epochs**:
  The automatically chosen number of epochs is 31 in my case. My strategy is taking the average of last 5 loss, and checking if new loss smaller from the last 5 losses average in some amount(0.0001 is my epsilon).
- **The plot of the training mean-squared error loss over epochs:**



Training mean-squared error loss

- **The plot of the validation 12-margin error over epochs:**



Plot of the Validation 12-margin Error

- **5 qualitative results on the validation set:** Results of my configuration on validation set is given below.

Results on the Validation Set

- **Discussion:**
  Our model is designed very simple and low structured with 2 layer and 16 kernel number, and we are not using any high level regularization or any kind of high level computations in our layers. As a result of these, we can conclude some advantages and disadvantages as below:
  Advantages:
  As we said, it is very simple and low level DNN, it is working very fast when we compare the high level DNN in both training phase and both testing phase. If the accuracy is not our main aim and we want to complete our processes fast enough, our model can be used. Because it's accuracy may be good enough for some sort of tasks
  Disadvantages:
  It is very hard to train due to lack of parameters and it's uncomplicated design. Therefore, if we seeks for more accurate results, we may dont want to use our model.

  Potential Ways to Improve:
  To improve our model, first we can construct more layered network. This provides us more flexibility on weights, and as layer count increased we can add more activation functions and more arrangements between layer.

Second, we can increase the kernel size, and we can vary the size between layers. This provides us more weights to train and more accurate results depends on these weights. Also by arranging different kernel sizes between layers, we can arrange our network to our specific task, and it will provide us more options to continue with.
Third, we can add fully connected layer in our network which provides us higher representations of images.
Besides these, we do not have time problem in our network for now, but after implementing the parts we discussed above we may need some performance in terms of time, so we can do pooling to decrease size at the end.

4 Your Results on the Test Set (30 pts)

To run the code, first set all the hyper-parameters on top of the code to given values, and than select Train as true for training and Test as true for Testing. After setting all parameters we can run the code by just using python model_code.py.

5 Additional Comments and References

To sum up, I have done many experiments with my model on difference configurations, and I have tried some further more experiments on it. Overall, I have learned that some changes in hyper-parameters does not have to effect our network in expected way especially when we working with uncomplicated simple network. Therefore, we should always consider different cases that hyper-parameters can cause in our models.