# CENG 483

## Introduction to Computer Vision

Fall 2023-2024

## Take Home Exam 3: Image Colorization

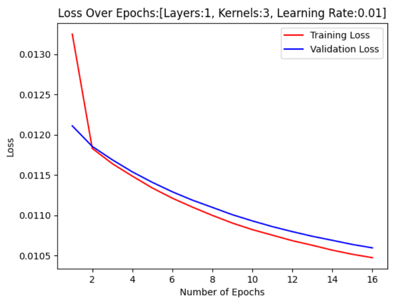
Full Name: Aybüke Aksoy
Student ID: 2448090

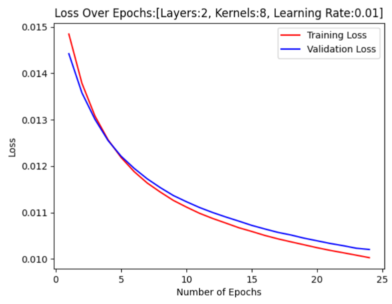
## 1 Baseline Architecture (30 pts)

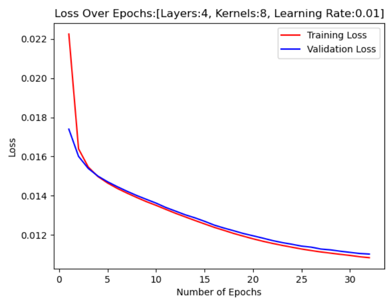
For this section, the training process is done with MME Loss function.

### 1.1 Effect of The Number of Convolution Layers

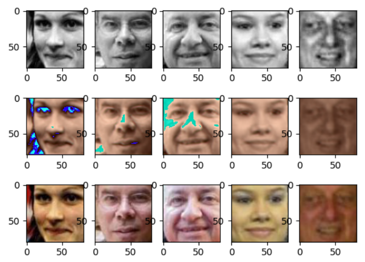
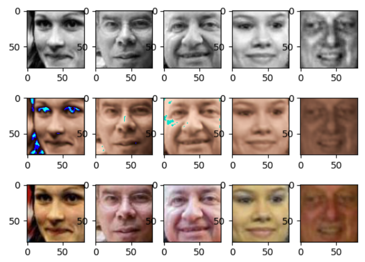
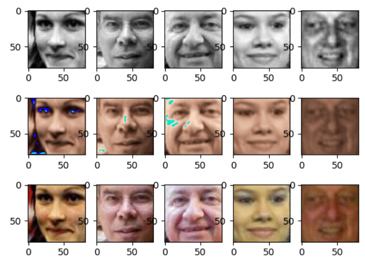
In this part, we tried to observe the effect of the number of convolution layers on the network and loss by keeping the other parameters such as number of kernels and learning rate constant. For 3 Kernels and 0.01 Learning Rate, the loss plots for each layer count over epochs are as:







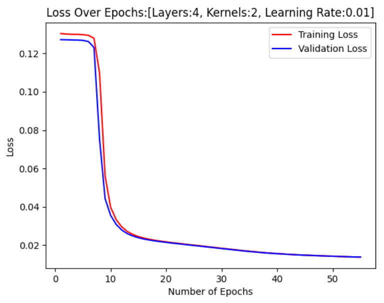
Generally, increasing the number of convolution layers increases the capacity to capture details and complex patterns but it can also cause overfitting which results in worse performance on unseen data.
The usage depends on the other parameters,the dataset and task complexity. If the dataset is small and consists of not very complex images, then more layers could be harmful as it is computationally costly and could result in overfitting but also small number of layers like 1 might limit the model's learning capacity which causes low accuracies.
Looking at our results, we can see that all nets converges to a relatively close loss points in different epoch counts. The net with 1 layer converges in approximately 16 epochs, the net with 2 layers converges in 24 epochs and the net with 4 layers converges in 33 epochs. From here, we can infer that deeper networks initially might take longer to train and the initial loss might be higher due to complexity. However, also considering the scales of the graphs, the net with 4 layers actually creates a smoother loss decrease. This shows better and more stable learning curve and convergence over good number of epochs. Also, I do not think that 4 layers cause overfitting for this dataset, as the performance on training and validation set are similar.

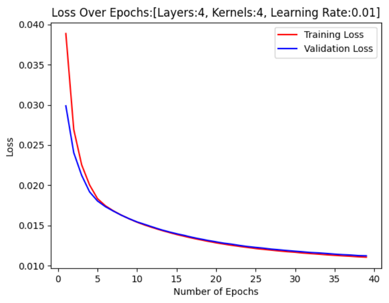




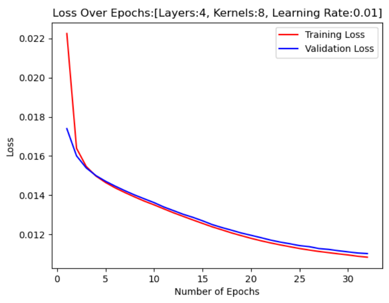
On the qualitative images from our dataset for each layer count, we can see that there are some extremely miscolored parts such as eyes, nose and background. However, as the layer number increases, this miscolorization decreases in considerable amount. This is due to deeper network's ability to capture more complex and rare occurrences.

## 1.2 Effect of The Number of Kernels

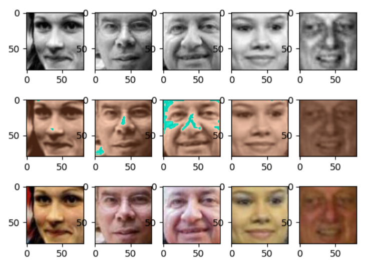
In this part, we tried to observe the effect of the number of kernels on the network and loss by keeping the other parameters such as number of convolution layers and learning rate constant. For 4 layers and 0.01 Learning Rate, the loss plots for each kernel count over epochs are as:

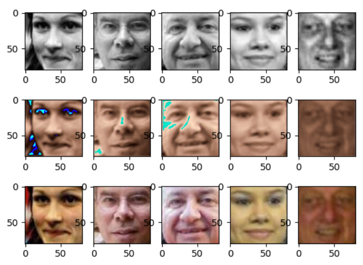


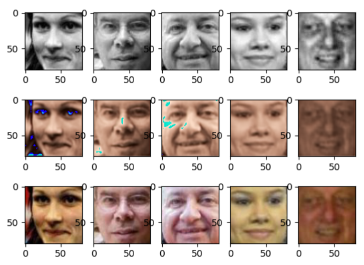




The number of kernels in a convolutional layer is related to the number of channels in the output feature map. Each kernel detects different patterns in the input data. Hence, the main effect of kernel number is on feature representation. To capture a diverse set and wider variety of features from the input images, a larger number of kernels might be needed. As in the convolution layer case, increasing the number of kernels is more suitable for complex tasks and datasets. Yet, this might also cause overfitting and increase in computational cost.
Looking at our results for each kernel number, we can again see that each net converges to a relatively close loss points in different epoch counts. The net with 2 kernels converges in approximately 56 epochs, the net with 4 kernels converges in 38 epochs and the net with 8 kernels converges in 33 epochs. We can infer that the network with less kernel number takes longer to converge. This is mainly due to our observation being on 4 layered network. As we know, deeper layers require more kernels to represent the distinct features it detects hence more kernels for this deep network creates smoother and more stable learning curve without drastic changes or drops in loss. Again, I do not observe any overfitting for this dataset, as the performance on training and validation set are close to eachother.



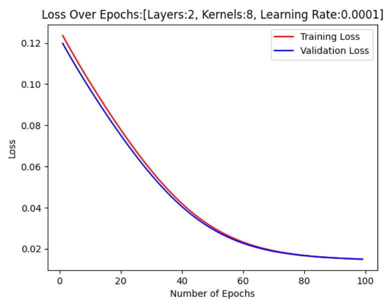


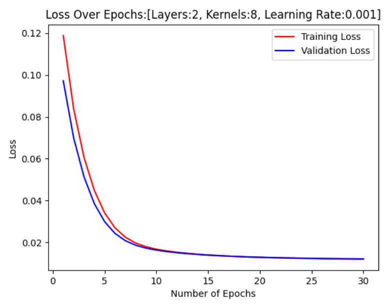


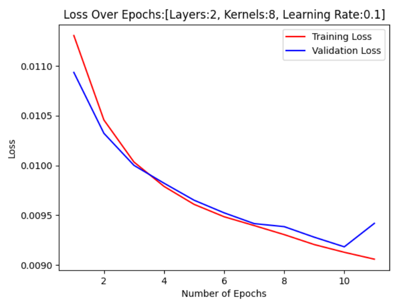
On the qualitative images from our dataset for each kernel count, we can see that there are some extremely miscolored parts in the middle image. Yet, as the kernel number increases, this blue miscolorization decreases in considerable amount and results in a better accuracy. This is due to higher number of kernels' ability to represent more complex and rare occurrences.

## 1.3 Effect of The Learning Rate

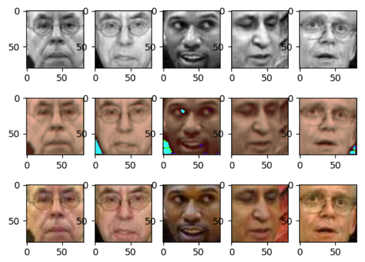
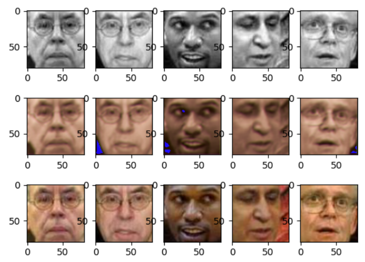
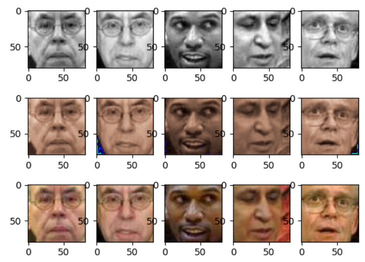
In this part, we tried to observe the effect of the learning rate on the network and loss by keeping the other parameters such as number of kernels and number of layers constant. For 8 Kernels and 0.01 Learning Rate, the loss plots for each layer count over epochs are as:







One of the most crucial hyperparameters while training the network is learning rate as it determines the convergence of the loss function. A very small learning rate such as 0.0001 leads to small updates to the parameters of the network in each iteration, resulting in reaching a minimum of the loss function in slowly. In the first plot, it can be seen that the iterations go until epoch=100 and we do not have information on where it converges. However, a very large learning rate as 0.1 leads to large updates to the parameters of the network in each iteration. In the last plot, it can be seen that he iterations stopped around epoch=10. The net might overshoot the minimum of the loss function due to this large steps and might fail to converge. The loss function values oscillates and shows unstability. Sometimes, it can even lead to divergence, and loss starts increasing instead. To observe those changes,detect the minimum points and stop before loss gets high, we are monitoring the training loss and validation loss. According to some condition based on the loss, which will be explained later, we are implementing an early stop to prevent this problem. In the middle plot, I have chosen an average learning rate such as 0.01. It can be seen that it converges at epoch=20 and is stable. Learning rates around 0.001-0.01 could overcome the problem of overshooting while also converging quickly.

On the qualitative images from our dataset for each learning rate, we can see that there are some miscolorization on the backgrounds. It is noticeable for the small learning rate 0.0001. This happens since we have the limit of 100 epochs for training. Model stops before the convergence point even if it still has the capacity to learn and could perform better if it was trained for a longer period.

## 2 Further Experiments (20 pts)

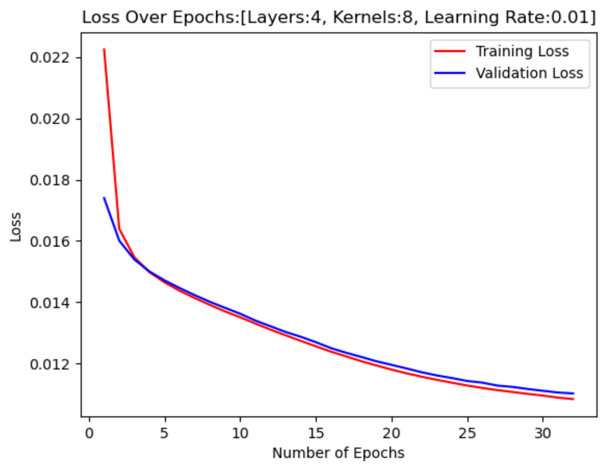
For this section of the experiment, I have chosen the hyperparameters such that and the networks are trained with MME Loss function:

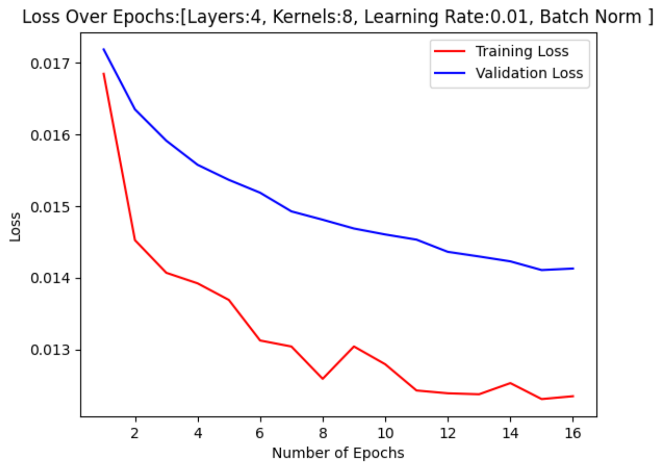
Number of Layers=4
Number of Kernels=8
Learning Rate=0.01

according to my observations from the loss plots and the reasons I have discussed above. The changes and additions will be made on the model using those hyperparameters.

### 2.1 Adding a Batch-Norm Function

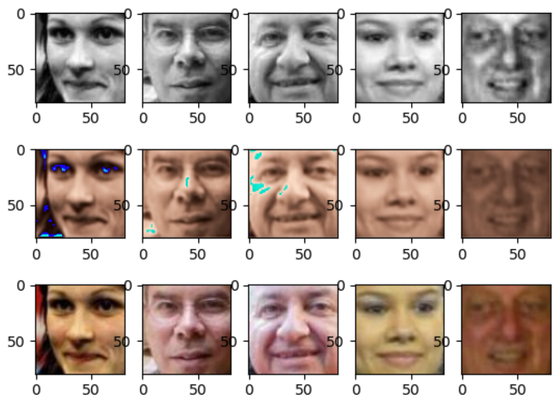
In this part, we tried to observe the effects of adding a Batch-Norm function to the each layer of the network, on loss and training by keeping the other parameters such as number of kernels, number of layers and learning rate constant. For previously mentioned hyperparameters, the results are as:

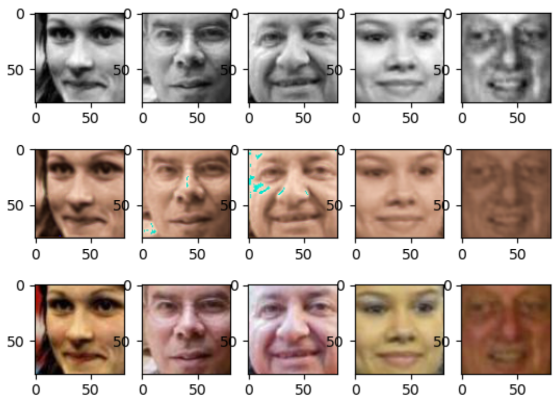




Batch normalization is a well known technique that is used to train the networks faster and more stable through normalization of the layers' inputs by reducing the internal covariant shift due the randomness in

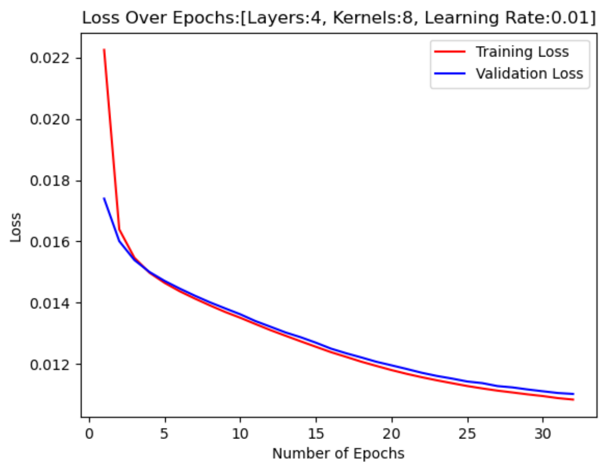
the parameter initialization and the randomness in the input data. It is known to have many other benefits in terms of convergence, adaptability etc. However, it may not always lead to a better performance and even degrade it.
In our case, by looking at the graphs, we can observe that the network with the batch-norm layer added right before ReLu function, has an unstable and oscillating loss function. Also, the difference between the training loss and the validation loss is extremely large with earlier stop. This might be due to learning rate increasing with batch normalization. With larger steps, it either converges earlier or diverges. It is hard to comment on that from this graph, however due to the increase in validation loss, we can easily say that out network did not benefited from the batch-norm layer.
The negative effects of batch-norm function might be due to various reasons such as incorrect placement in the network or learning rate sensitivity. We could try to change the momentum and epsilon parameters of the batch-norm function, try placing it after the ReLu function or changing the batch size to see if we can benefit under any configuration. I have not implemented those and hence decided not to add the batch-norm layer.

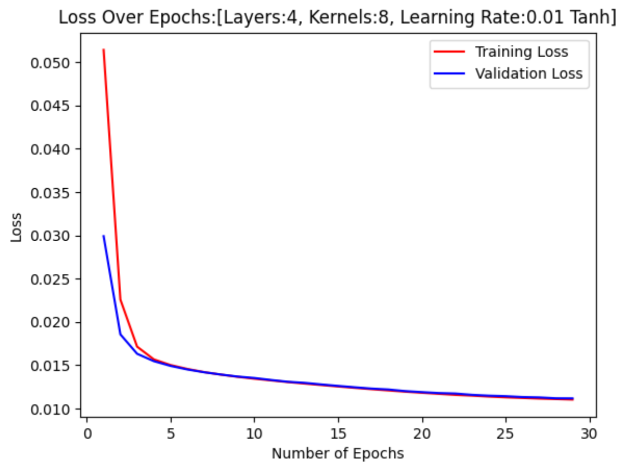




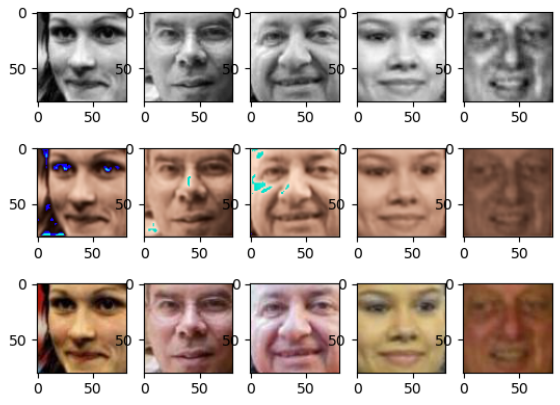
## 2.2 Adding a Tanh Activation Function

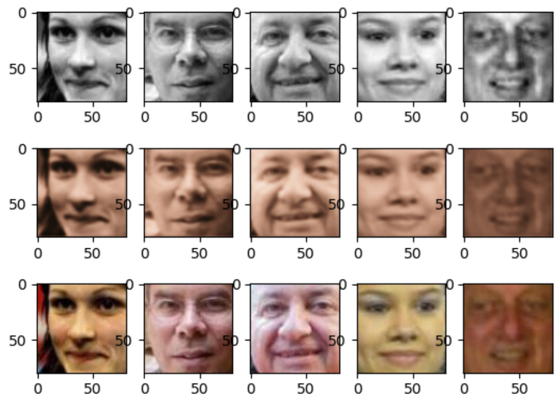
In this part, we tried to observe the effects of adding a Tanh activation function to the last layer of the network, on loss and training by keeping the other parameters such as number of kernels, number of layers and learning rate constant. For previously mentioned hyperparameters, the results are as:





The Tanh function is a mathematical function that maps real-valued numbers to the range [-1, 1]. In our implementation, the network takes grayscale intensities normalized to the range between -1 and +1 as input and it output R, G and B channels for the color image in size again in the range from -1 to +1 for each pixel color value. Since Tanh activation function squashes the input values to this range, it might be beneficial in cases where output needs to be normalized or predictions has to be in a specific range. This idea seems suitable to our implementation also it is said that tanh function can be useful in capturing increasing and decreasing patterns in the data. It seems suitable for our implementation due to the range constraints we have.
Looking at the plots, there is not much difference between the validation loss on the converge points, but the one with tanh activation function converges faster, and there are sudden decreases in the loss during the first steps.
This results made me hesitate and question whether tanh will be beneficial. Therefore, I have checked prediction and target images and compared them:



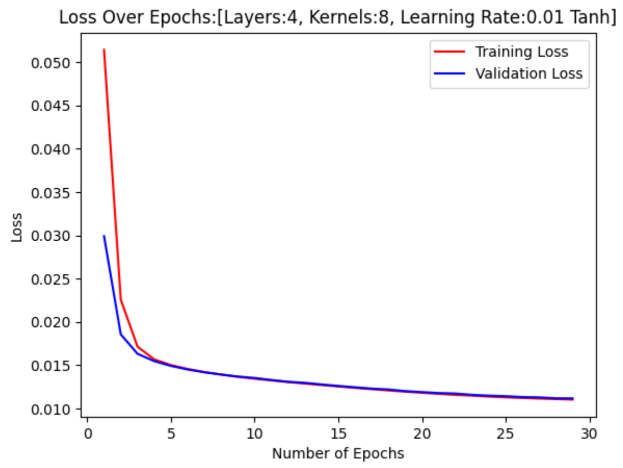


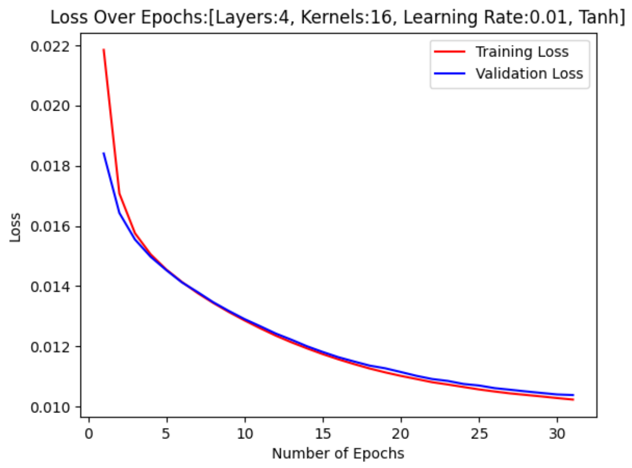
On the qualitative images from our dataset obtained with the net without tanh activation function, we can see that there are some miscolorization especially on the middle image. Yet, in the one obtained with the net that tanh is applied, those miscolorizations are not visible anymore and our predictions are closer to the targets. I think, the change in the loss and images are in contradiction and I do not know the exact reason for this. However, since I thought image results were more important, I decided to add

the tanh activation function to the network.
The effect of it will be explained in additional comments part since I have done some additional tests and observations and realized that adding it was not a good choice.

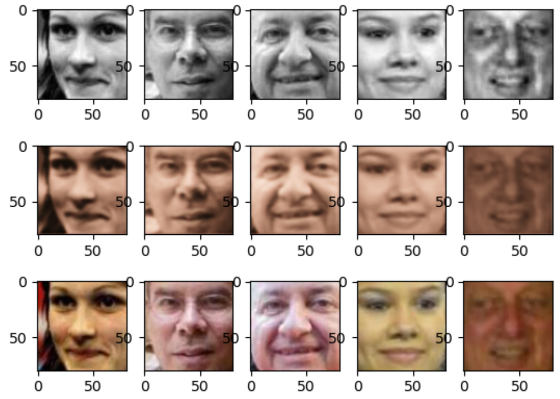
## 2.3 Setting The Number of Channels to 16

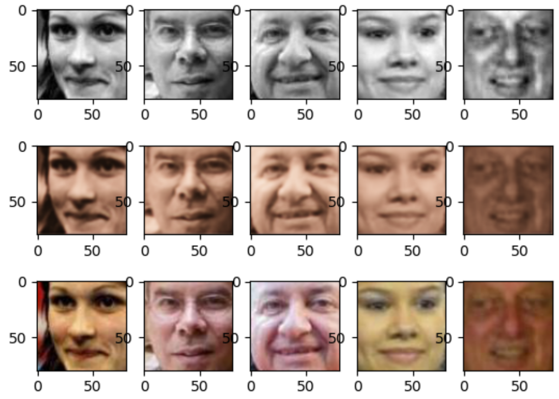
In this part, we tried to observe the effects of increasing the number of kernels on loss and training by keeping the other parameters such as number of layers and learning rate constant. For previously mentioned constant hyperparameters and the updated channel number, the results are as:





The number of channels, which are kernels in a convolutional layer is related to the number of channels in the output feature map. Each channel detects different patterns in the input data as it is mentioned in the first section. Higher number of channels enable the capture of more diverse and wider variety of features with higher computational cost and possibility of overfitting. Looking at our results for kernel number 8 and 16, we can again see that each net converges to a relatively close loss points in close epoch counts. The net with 8 kernels converges in approximately 28 epochs and the net with 16 kernels converges in 31 epochs. As also mentioned before, deeper layers require more kernels to represent the distinct features it detects hence more kernels for this deep network creates smoother and more stable learning curve without drastic changes or drops in loss. The second plot is in aliance with this idea. Again, I do not observe any overfitting for this dataset, as the performance on training and validation set are close to eachother.
For the reasons above, I have decided to change the number of channels to 16.





On the qualitative images from our dataset obtained with the net with 16 channels and 8 channels, we can see that there is not any observable difference between, and both are accurate.

## 3 Your Best Configuration (20 pts)

Considering the results I have obtained until now, I have decided to use the network with the hyperparameters:

Number of Layers=4
Number of Kernels=16
Learning Rate=0.01
Batch-Norm: No
Tanh: Yes

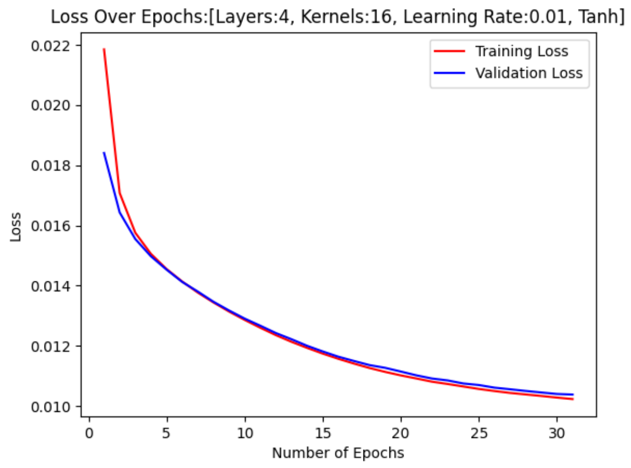
While choosing a good architecture, I have decided to try the layer numbers 1,2,4; the kernel numbers 2,4,8; and the learning rates 0.0001, 0.001, 0.01,0.1. I have done grid search with all the possible combinations of those (there were 28 as kernel for layer 1 is only 3) to find a good enough architecture. Then I have added 2 more parameters and tried the combinations of adding batch-norm and tanh parameters on the architecture I have chosen.
While training the network and tuning the hyperparameters, I have considered both MSE Loss and 12-Margin Loss. The previous loss plots were with respect to MSE Loss, however in this section 12-Margin Loss will be more crucial. For the best configuration, the results for models trained with both loss functions will be shown.

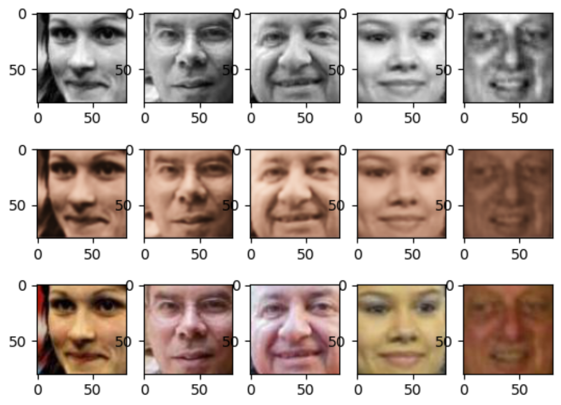
### 3.1 Number of Epochs

MSE Loss: 31
12-Margin Loss: 23

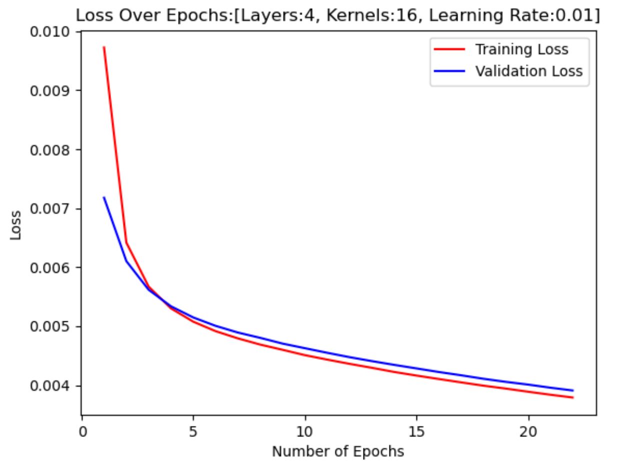
In the previous parts, it was mentioned that an early stop method was implemented depending on some conditions. In my early stop implementation, which stops the training after learning slows down or in case of divergence, depends on the last 3 validation loss values. The average of these values are taken and it is checked whether the new value for validation loss is greater than that average with epsilon 0.00001. If it is greater, the training stops, as it means that loss is increasing. This increase shows that either the loss function converged and no more training is required, or it overshooted the minimum point and loss will get worse from now on. In each case, there is no need to continue.

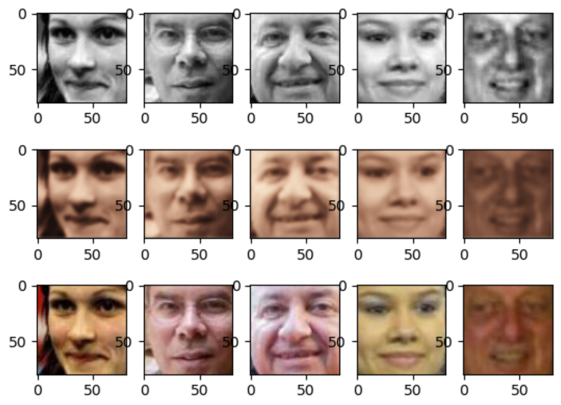
## 3.2 The Plot of The Training Mean-Squared Error Loss Over Epochs





## 3.3 The Plot of The Training 12-Margin Error Loss Over Epochs





## 3.4 Discussion

Throughout the experiement, we tried to tune the network with various configurations for hyperparameters. As the experiements was done step by step and in a hierarchial manner, the actual best configuration can be different than what I chose by considering the loss plot results and the changes on the prediction images.
My best configuration has 4 layers, which is computationally expensive and takes time to train, yet it is better at capturing the complex images. This also could be chosen as 2 layers (this gives good results as well). That choice would still be accurate enough and less computationally expensive. Number of kernels is chosen as 16 which helps to represent the distinct features that we have captured with the 4 layers. Learning rate is 0.01 and this was chosen between 4 options that I have tried. Maybe a value like 0.05 could perform better if that was implemented. A better option can be using an adaptive learning rate algorithms, such as Adam or Adagrad. Different optimization functions could be tried instead of Tanh such as Swish or Parametric Rectified Linear Unit (PReLU). Also applying max pooling could reduce the

computational complexity and memory usage.
Overall, I think the model is accurate enough in image colorization but it can be seen that it lacks in variety of skin color etc when the predicted images are observed.

## 4 Your Results on the Test Set (30 pts)

For testing, *estimations_test.npy* array and *test_images.txt* is provided. To learn how to run the code, the related README file can be checked.

## 5 Additional Comments and References

After the experiments, to have an idea of my accuracy results, I have tried the evaluation on 100 images of validation set. For my chosen best configuration, the accuracy was %62. I have tried a few more configurations which resulted in better accuracy. When I discarded the Tanh function, the accuracy went up to %65. When I choose the layer number as 2, it went up to %70. I am only submitting the estimations.npy for my chosen best configurations, as that was what I decided on during the experiment process due to reasons explained. The test was done later, and the actual accuracies are not certain, but if there is need to test them, the checkpoints with better accuracies are saved and can be shared.