CENG 483

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

Full Name: Derya TINMAZ
Student ID: 2380947

1 Baseline Architecture

Throughout the hyperparameter tuning process for image colorization, I maintained a consistent configuration across all models, setting a batch size of 16 and a kernel size of 3 in the convolutional layers, with each layer (except the final one) followed by a ReLU activation function. These models underwent a thorough testing regime over 50 epochs. The patience parameter is used to implement early stopping. When the validation loss fails to show any improvement over 10 (two 5 epochs) epochs, the training process is halted prematurely to prevent overfitting. For a detailed performance evaluation, I conducted a comprehensive analysis of the validation set every 5 epochs to calculate the average validation loss. The report includes train loss vs. validation loss graphs and 12-error margin and accuracy score graphs on every 5 epochs for each model, providing a clear visual representation of their performance and learning trends over the training period.

1.1 The Number Of Convolutional Layers

In examining the impact of the number of convolutional layers on model performance, I conducted tests on three different models, each varying in the number of convolutional layers. The common parameters for these models were a fixed number of kernels set at 2 and a learning rate of 0.1. The distinguishing factor among the models was the number of convolutional layers, with the models featuring 1, 2, and 4 layers respectively.

number of conv layers	train loss	validation loss	12-margin error	accuracy
1	0.0091	0.0091	0.2466	0.7534
2	0.0085	0.0086	0.2341	0.7659
4	0.1308	0.1272	0.7961	0.2039

The model with a single convolutional layer showed moderate effectiveness, reflected in train and validation losses of 0.0091 and 0.0091, a 12-margin error of 0.2466, and an accuracy of 75.34%. An improvement was observed with two layers, evident from lower train and validation losses (0.0085 and 0.0086), a reduced 12-margin error of 0.2341, and increased accuracy at 76.59%. However, increasing to four layers led to much higher train and validation losses (0.0108 and 0.1308), a larger 12-margin error of 0.1272, and decreased accuracy to 20.39%. This suggests that while adding more layers can enhance the model's capability to a certain extent, excessive layers may lead to overfitting and reduced overall performance drastically. The

optimal balance of depth and performance in this context appears to be achieved with two convolutional layers. Therefore, for the next models I used two convolutional layers

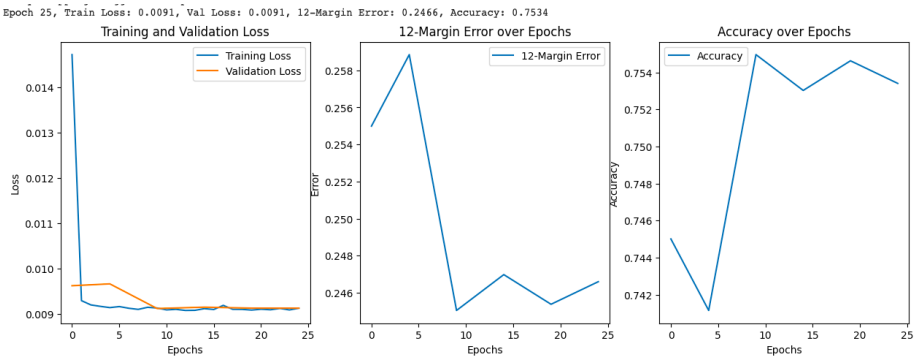

Figure 1: **model-1:** 1 convolutinoal layer

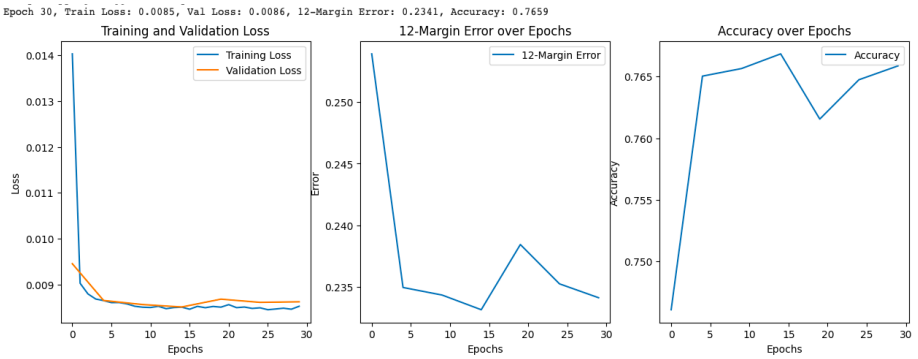

Figure 2: **model-2:** 2 convolutinoal layers

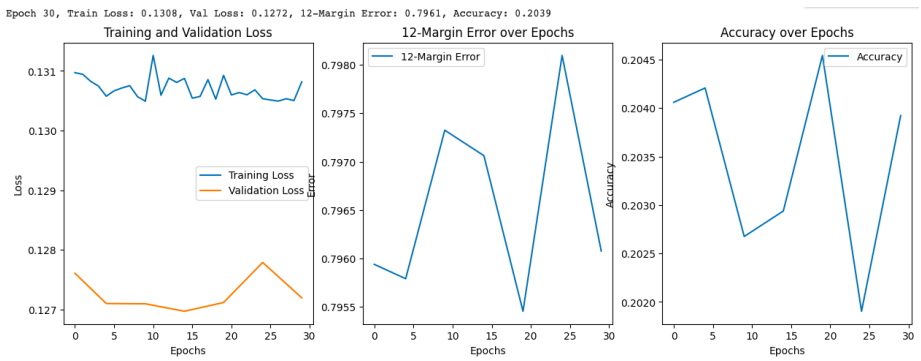

Figure 3: **model-3:** 4 convolutinoal layers

1.2 The Number Of Kernels

In examining the impact of the number of kernels on model performance, I conducted tests on three different models, each varying in the number of kernels. The common parameters for these models were a fixed number of convolutional layers set at 2 and a learning rate of 0.1. The distinguishing factor among the models was the number of kernels, with the models featuring 2, 4, and 8 kernels respectively.

number of kernels	train loss	validation loss	12-margin error	accuracy
2	0.0085	0.0086	0.2341	0.7659
4	0.0083	0.0084	0.2319	0.7681
8	0.0083	0.0084	0.2280	0.7720

The first model, with 2 kernels, showed reasonable effectiveness, indicated by train and validation losses of 0.0085 and 0.0086, a 12-margin error of 0.2341, and an accuracy of 76.59%. A slight improvement was observed with 4 kernels, which reduced the train and validation losses to 0.0083 and 0.0084, decreased the 12-margin error to 0.2319, and increased accuracy to 76.81%. The most notable enhancement came with the model featuring 8 kernels. It achieved the best results with the lowest train and validation losses (0.0083 and 0.0084), a 12-margin error of 0.2280, and the highest accuracy at 77.20%.These results suggest that increasing the number of kernels up to a certain point can positively impact the model's performance, likely due to the enhanced ability to capture more complex features and patterns in the image data. Therefore, for the next models I used eight kernels.

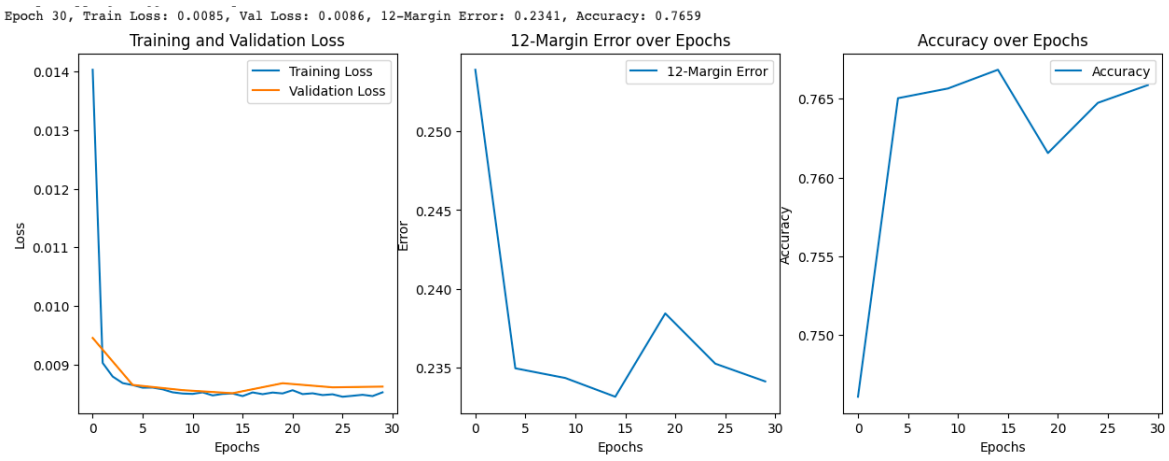

Figure 4: **model-4 (same with model-2):** 2 kernels

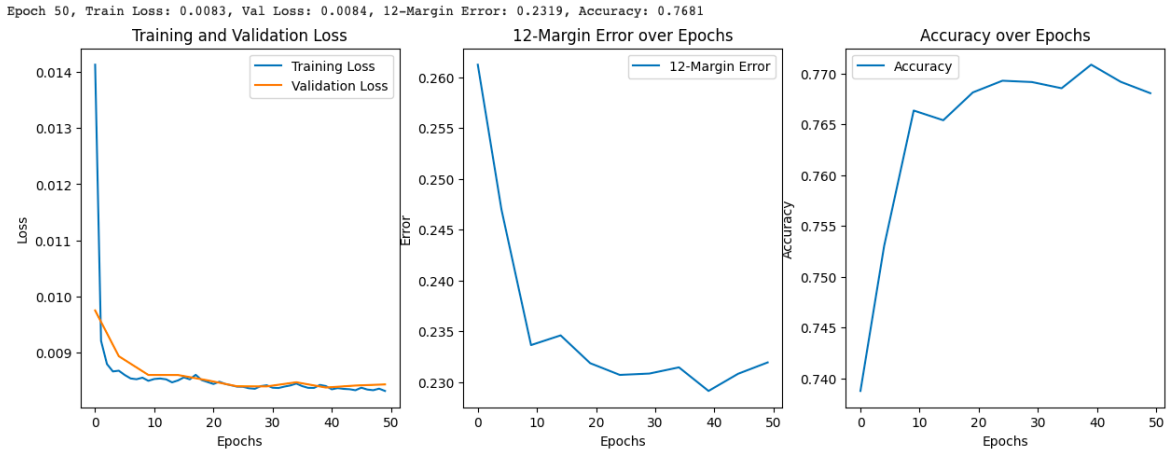

Figure 5: **model-5:** 4 kernels

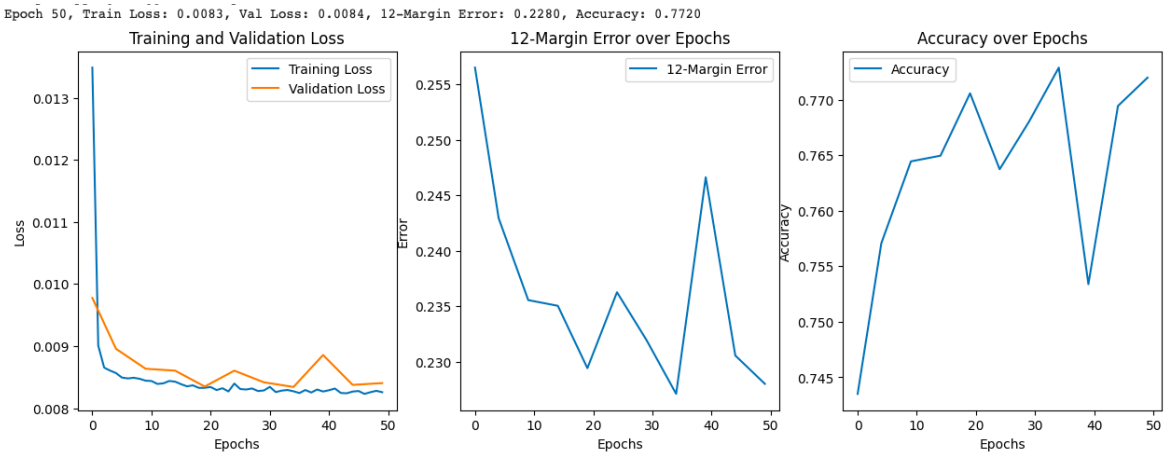

Figure 6: **model-6:** 8 kernels

1.3 The Learning Rate

In examining the impact of the learning rate on model performance, I conducted three different tests, each varying in the learning rate. The common parameters for these models were a fixed number of convolutional layers set at 2 and fixed number of kernels set at 8. The distinguishing factor among the models was the learning rates, with the models featuring 0.1, 0.01, and 0.001 learning rate respectively.

learning rate	**train loss**	**validation loss**	**12-margin error**	**accuracy**
0.1	0.0083	0.0084	0.2280	0.7720
0.01	0.0083	0.0084	0.2294	0.7706
0.001	0.0095	0.0097	0.2565	0.7435

The highest learning rate of 0.1 demonstrated the best performance, with a train loss of 0.0083, a validation loss of 0.0084, a 12-margin error of 0.2280, and an accuracy of 77.20%. When the learning rate was reduced to 0.01, the train and validation losses remained the same, but there was a slight increase in the

12-margin error to 0.2294 and a marginal drop in accuracy to 77.06%. A further reduction of the learning rate to 0.001 resulted in significantly worse outcomes: increased train and validation losses (0.0095 and 0.0097), a higher 12-margin error of 0.2565, and reduced accuracy to 74.35%. These findings suggest that a higher learning rate of 0.1 is more effective for this task, likely due to faster convergence and better optimization, while lower rates lead to suboptimal learning and reduced model accuracy. Therefore, I used this learning rate for the subsequent models.

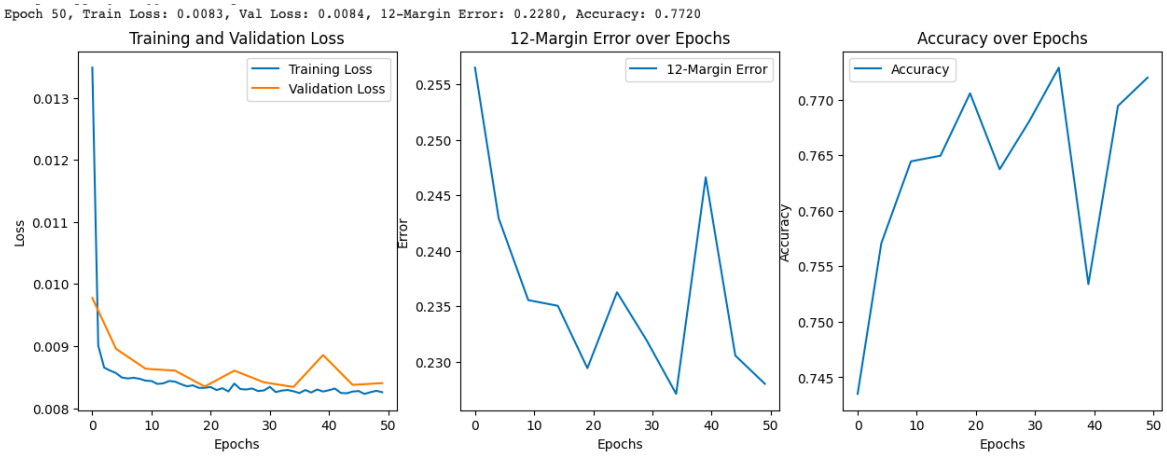

Figure 7: **model-7 (same with model-6):** 0.1 learning rate

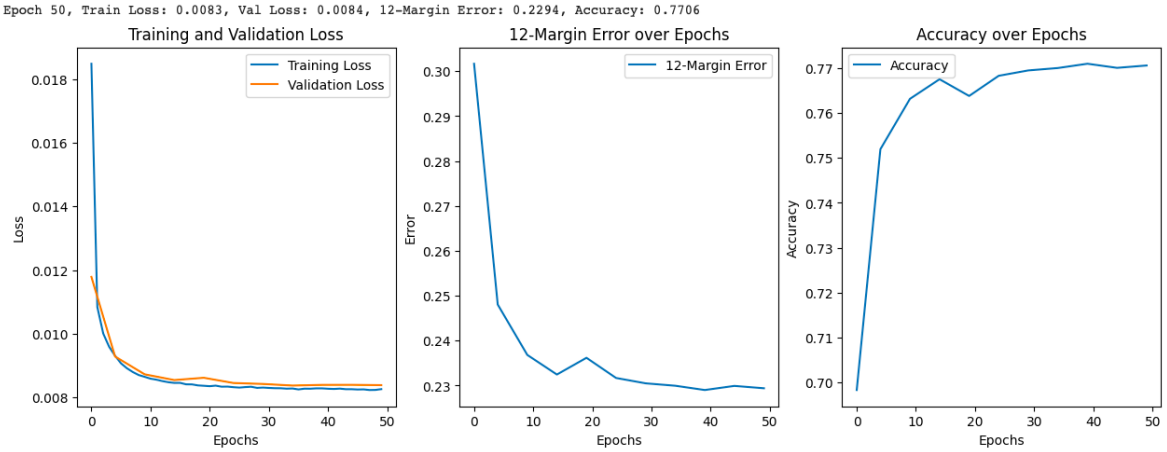

Figure 8: **model-8:** 0.01 learning rate

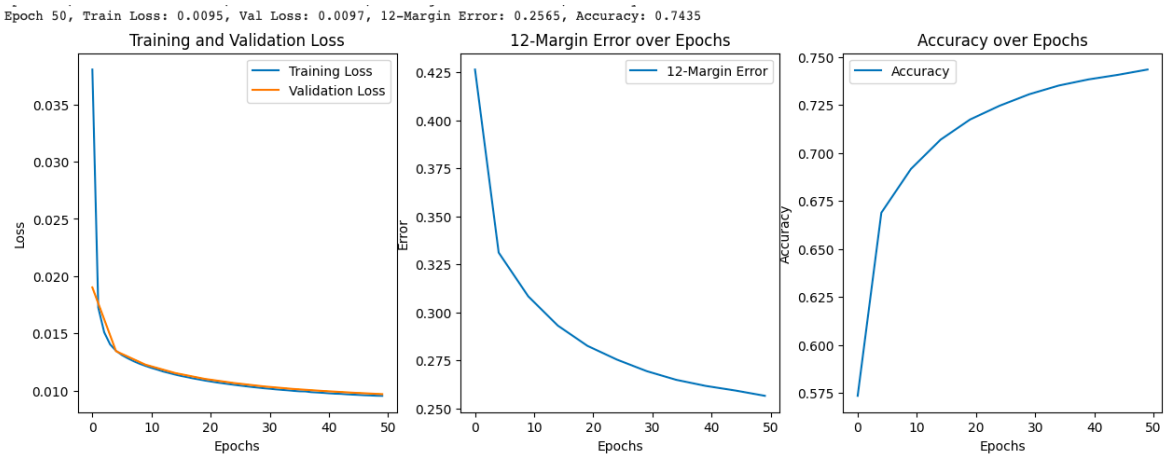

Figure 9: **model-9:** 0.001 learning rate

2 Further Experiments

2.1 Batch-Norm

To evaluate the impact of batch normalization, I utilized the best-performing configurations from previous tests as a baseline: 2 convolutional layers, 8 kernels, and a 0.1 learning rate. On top of this, I integrated a batch normalization layer into each convolutional layer for comparison.

model	train loss	validation loss	12-margin error	accuracy
without batch-norm	0.0083	0.0084	0.2280	0.7720
with batch-norm	0.0110	0.0088	0.2456	0.7544

The model without batch normalization showed a train loss of 0.0083, a validation loss of 0.0084, a 12-margin error of 0.2280, and an accuracy of 77.20%. In contrast, the introduction of batch normalization resulted in a higher train loss of 0.0110, a slightly higher validation loss of 0.0088, a marginally increased 12-margin error of 0.2456, and a lower accuracy of 75.44%. These results suggest that batch normalization, typically used to stabilize and accelerate training, did not yield a improvement in this context. Given that the data was already normalized, the batch normalization layers might have introduced redundancy or unnecessary complexity, which slightly impeded the model's performance. Therefore, it seems more beneficial to proceed without batch normalization in this setup.

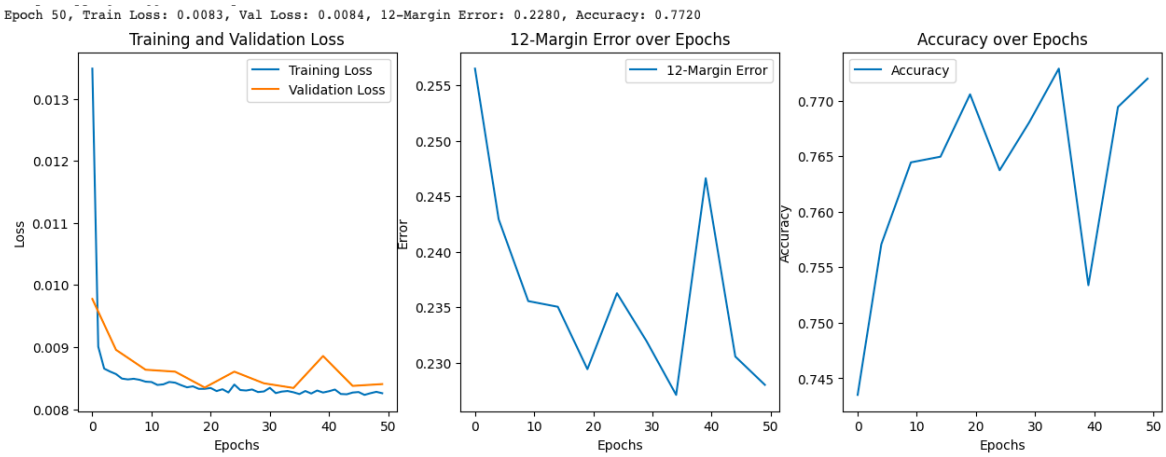

Figure 10: **model-10 (same with model-6 & model-7):** without batch-norm

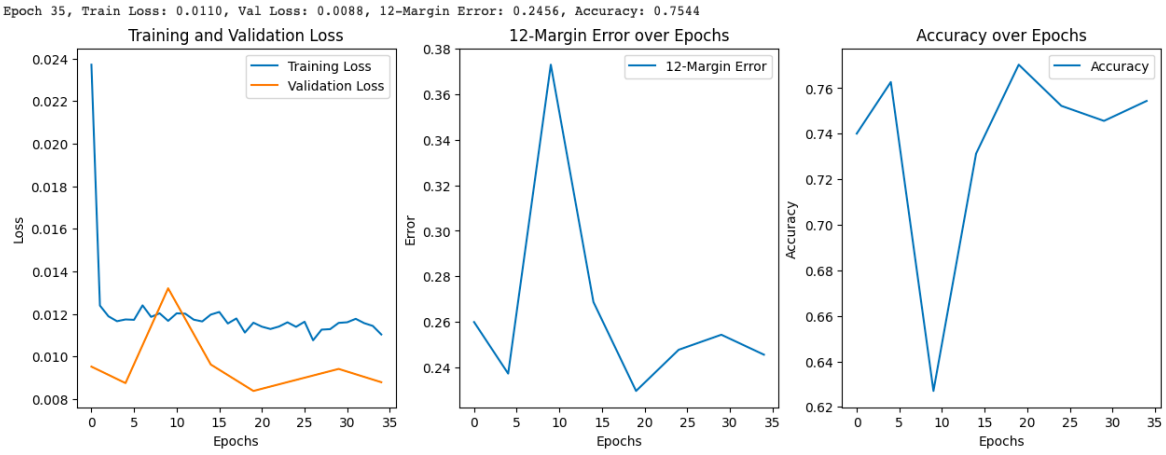

Figure 11: **model-9:** with batch-norm

2.2 Tanh

To evaluate the impact of tanh activation function , I utilized the best-performing configurations from previous tests as a baseline: 2 convolutional layers, 8 kernels, and a 0.1 learning rate without batch-norm. On top of this, I integrated a tanh activation function after the very last convolutional layer for comparison.

model	train loss	validation loss	12-margin error	accuracy
without tanh	0.0083	0.0084	0.2280	0.7720
with tanh	0.0083	0.0084	0.2328	0.7672

The original model without tanh achieved a train loss of 0.0083, a validation loss of 0.0084, a 12-margin error of 0.2280, and an accuracy of 77.20%. The addition of the tanh function resulted in the same train and validation loss, but with a marginally increased 12-margin error of 0.2328 and a lower accuracy of 76.72%. These results indicate that while the tanh activation function aligns with the data normalization

range, its integration at the end of the model did not significantly enhance performance and marginally reduced the accuracy. This suggests that the model's existing setup was already well-optimized for the task, and the addition of tanh offered no substantial benefit, thereby making it more advantageous to continue with the configuration excluding the tanh activation function.

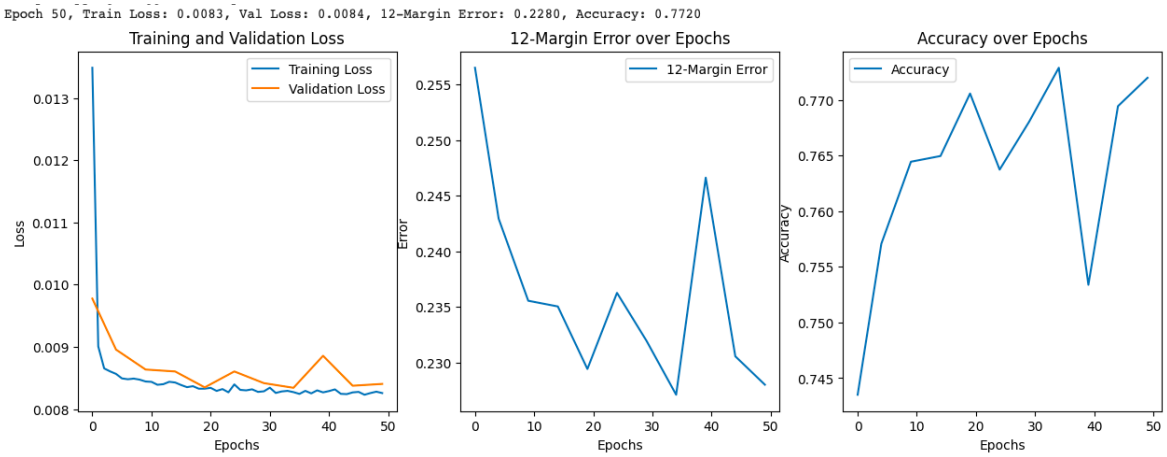

Figure 12: **model-12 (same with model-6 & model-7 & model-10):** without tanh

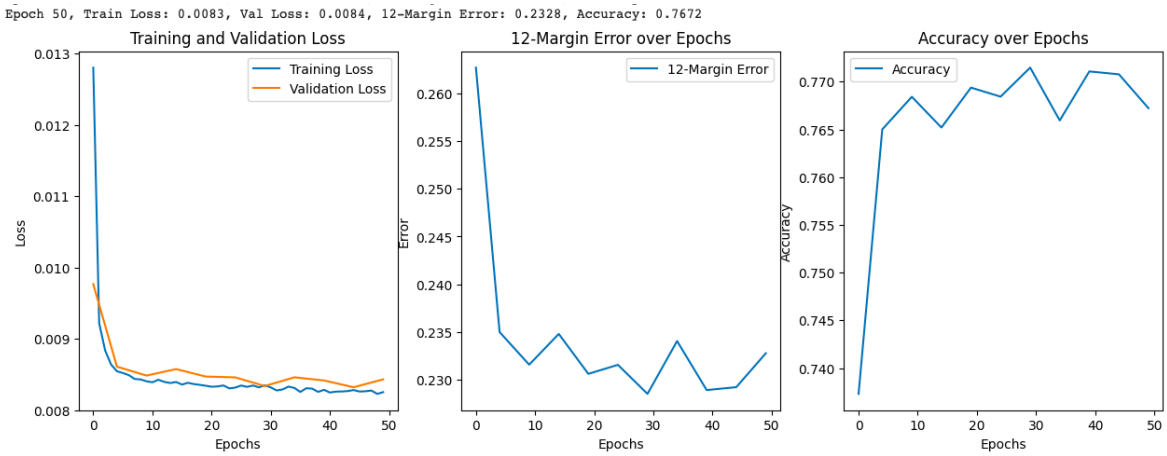

Figure 13: **model-13:** with tanh

2.3 16 Channels

To evaluate the impact of 16 channels , I utilized the best-performing configurations from previous tests as a baseline: 2 convolutional layers, and a 0.1 learning rate. On top of this, I used 16 kernels for comparison.

channel number	train loss	validation loss	12-margin error	accuracy
8	0.0083	0.0084	0.2280	0.7720
16	0.0081	0.0081	0.2228	0.7772

The original 8-channel model yielded a train loss of 0.0083, a validation loss of 0.0084, a 12-margin error

of 0.2280, and an accuracy of 77.20%. However, the modified model with 16 channels showed a slight improvement in all metrics: the same train loss of 0.0081, a reduced validation loss of 0.0081, a lower 12-margin error of 0.2228, and a marginally increased accuracy of 77.72%. These findings suggest that increasing the number of channels to 16 can positively impact the model's performance, likely by enabling it to capture more detailed and nuanced features within the images. This enhancement in channel count appears to offer a subtle yet noticeable improvement in accuracy and error reduction, thus making it a beneficial modification for this specific image colorization task.

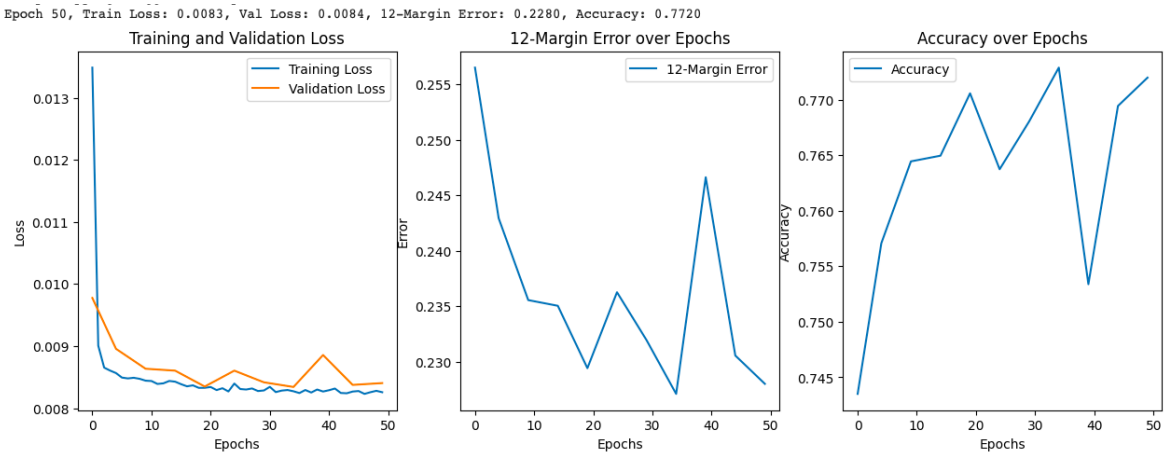

Figure 14: **model-14 (same with model-6 & model-7 & model-10 & model-10):** 8 channels

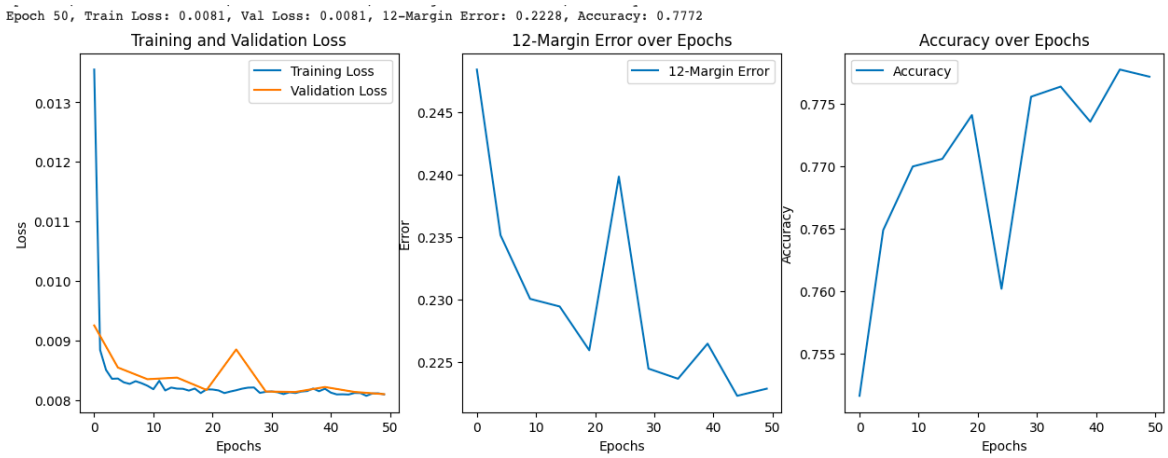

Figure 15: **model-15:** 16 channels

3 Best Configuration

Automatically Chosen Number of Epochs:
The patience parameter is used to implement early stopping. This mechanism is activated to closely monitor the validation loss at intervals of every five epochs, comparing it against the best validation loss observed to date. Whenever the validation loss in the current epoch undercuts the best recorded value, the function saves the state of the model and resets the counter that tracks epochs without improvement. If the validation loss fails to show any improvement over a stretch of epochs equal to the specified patience value, the training process is halted prematurely to prevent overfitting.

My Best Configuration: I trained my final model with the following configuration.

batch size	kernel size	kernel count	learning rate	conv layers	batch norm	tanh	max epochs
16	3	16	0.1	2	x	x	100

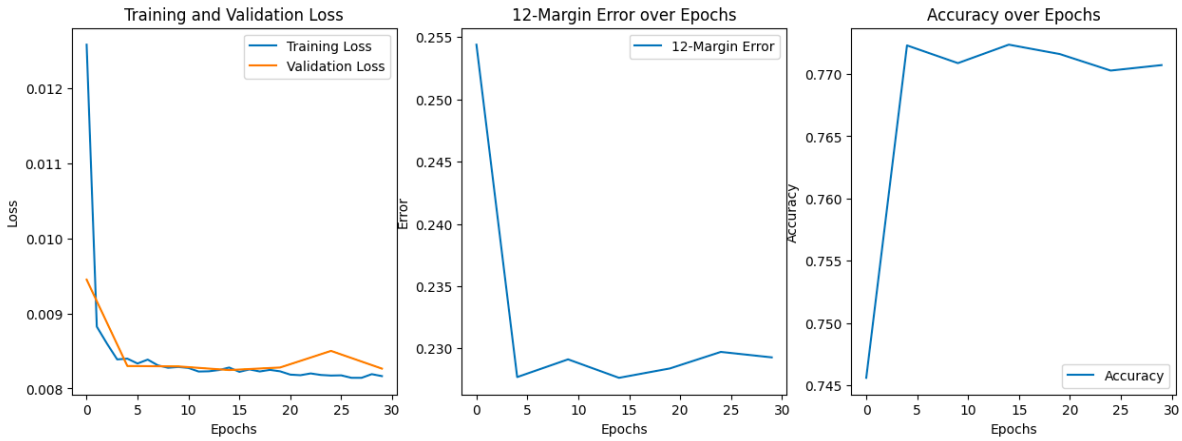

Figure 16: Graphs for the best configuration

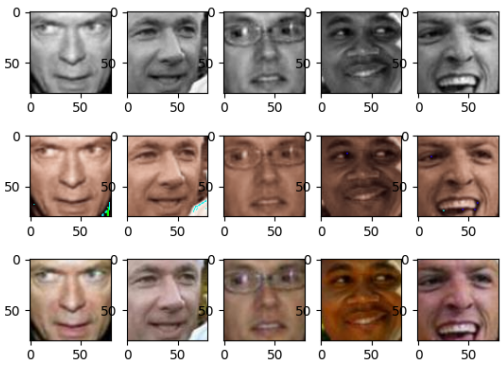

Figure 17: 1-input images, 2-predictions, 3-targets

Advantages of the Model

- The choice of 2 convolutional layers with 16 kernels each strikes a balance between model complexity and performance. It ensures sufficient feature extraction without overcomplicating the model, which is crucial for image colorization tasks.
- The learning rate of 0.1, which proved to be optimal in the experiments, facilitates efficient training. It's high enough to ensure fast convergence but not too high to cause instability in the learning process.
- The batch size of 16 and kernel size of 3 seem well-suited for the task. A moderate batch size helps in generalizing the model, while the kernel size allows for capturing the necessary spatial relationships in the images.
- Omitting batch normalization and the tanh activation function, based on the tests, suggests a more streamlined model architecture. This simplification can lead to faster training times and reduced model complexity.

Disadvantages of the Model

- Given the complexity of image colorization, there's a risk of overfitting, especially with a higher number of epochs (up to 100). Even though early stopping is implemented, careful monitoring of the validation loss is essential.
- While ReLU is a standard choice, exploring other activation functions could provide nuanced benefits for different layers of the model.
- The chosen hyperparameters, though optimal in the experiments, might not generalize well across different datasets or varying image resolutions.

Potential Improvements

- Implementing various data augmentation techniques could help in improving the generalization capabilities of the model. This can include rotations, scaling, or color adjustments.
- To further prevent overfitting, incorporating dropout layers or L1/L2 regularization in the neural network can be considered.
- Conducting further experiments with different combinations of hyperparameters, like varying the number of layers or adjusting the learning rate, can help in identifying an even more optimized model configuration.
- Experimenting with different loss functions, such as perceptual loss or a combination of losses, might yield more natural and visually pleasing colorization results.

4 Results on the Test Set

Since I used google colab, you can just run hyperparamater_tuning.ipynb and model.ipynb. In test_images.txt the path is the colab's path, if you test it in local, you can change the path accordingly. (I used first 100 images in the test data)

5 Additional Comments and References