Image Colorization

Ahmed Rabea, Marwan Mahmoud and Youssef Ashraf
Alexandria University

September 15, 2023

Abstract

Image colorization is the task of assigning colors to a grayscale image such that the resulting colored image is visually pleasing and semantically meaningful. We implemented the model proposed by "Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2"[1] paper and explored a few modifications to the architecture. We also explored transfer learning using different pre-trained models. Our results show that the model is able to successfully color high-level image components such as the sky, the sea, or forests. However, the performance in coloring small details is still to be improved. We believe that further improvements can be made by training the network over a larger training dataset.

1 Introduction

Image colorization is a challenging problem due to the varying conditions of imaging that need to be dealt with by a single algorithm. The problem is also severely ill-posed as two out of the three image dimensions are missing. While scene semantics can be helpful in many cases, such as knowing that grass is usually green, clouds are usually white, and the sky is usually blue, such semantic priors are not always available for man-made and natural objects, such as shirts, cars, and flowers. Additionally, the colorization problem inherits the typical challenges of image enhancement, such as changes in illumination, variations in viewpoints, and occlusions.

However, our goal is not to recover the actual ground truth color, but rather to produce a plausible colorization that could potentially fool a human observer. This makes our task much more achievable.

2 Approach

2.1 LAB Color Space

The CIE Lab color space separates the image information into three components: L (luminance), a (green to magenta), and b (blue to yellow). We chose the CIE Lab color space to represent the input images because it separates the color characteristics from the luminance, which contains the main image features.

The colorization task can be treated as a regression problem, where the goal is to predict the a and b channels for a given L channel. This simplifies the learning process for machine learning models, as it reduces the dimensionality of the output space compared to directly predicting RGB values.

2.2 Objective Function

The optimal model parameters are found by minimizing an objective function defined over the estimated output and the target output. The objective function is the mean squared error (MSE) between the estimated pixel colors in the ab space and their true values.

2.3 Data Preprocessing

To facilitate accurate learning, we applied a crucial preprocessing step to the dataset. Specifically, we center and scale the pixel values of all three image components. This transformation is employed to normalize the data and ensure that the pixel values fall within the interval of [-1, 1].

2.4 Baseline Architecture

The network is logically divided into four main components: the encoder, the feature extractor, the fusion layer, and the decoder. The encoding component and the feature extraction component acquire mid-level and high-level features, respectively. These features are subsequently merged within the fusion layer. Finally, the decoder uses these features to estimate the output.

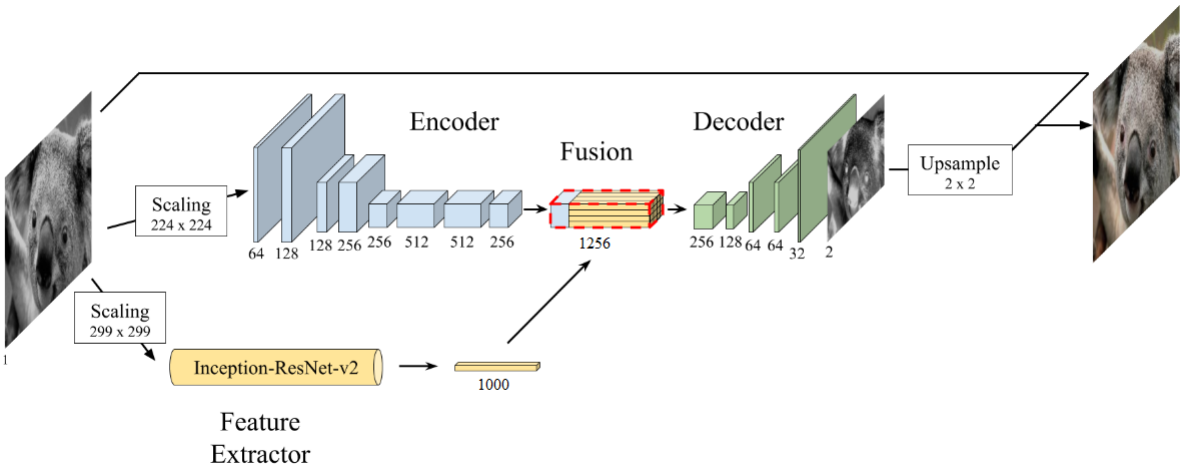
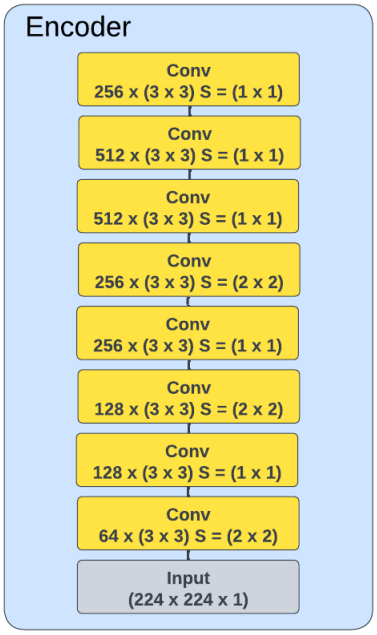


Figure 1: An overview of the baseline model architecture.

Encoder: The encoder takes a grayscale image of size H × W and outputs a feature representation of size H/8 × W/8 × 256. It does this by using 8 convolutional layers with 3 × 3 kernels. Padding is used to preserve the size of the input image at each layer.
Each convolutional layer uses a ReLU activation function. The first, third, and fifth layers use a stride of 2, which halves the size of the output from each layer. This helps to reduce the number of computations required.



Feature Extractor: The feature extractor extracts high-level features from the input image. These features can be used in the colorization process to represent the image's semantic content.

To extract the features, we use a pre-trained Inception model. First, we scale the input image to 299 × 299 pixels. Next, we stack the image (as shown in Fig. 2) with itself to obtain a three-channel image. This is done to satisfy the dimension requirements of the Inception model. Finally, we feed the resulting image to the network and extract the output of the last layer before the softmax function. This results in a 1000×1×1 embedding.

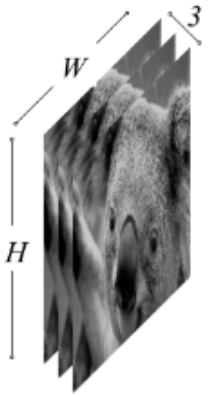


Figure 2: Stacking the luminance component three times

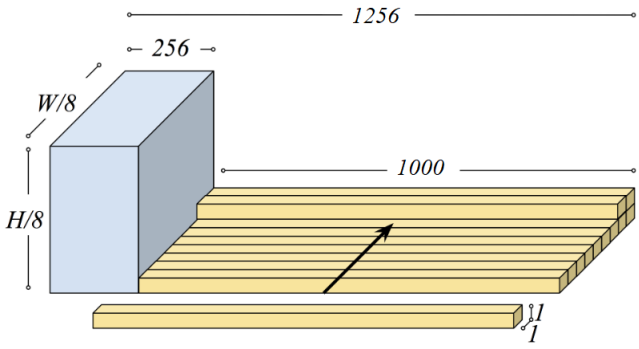
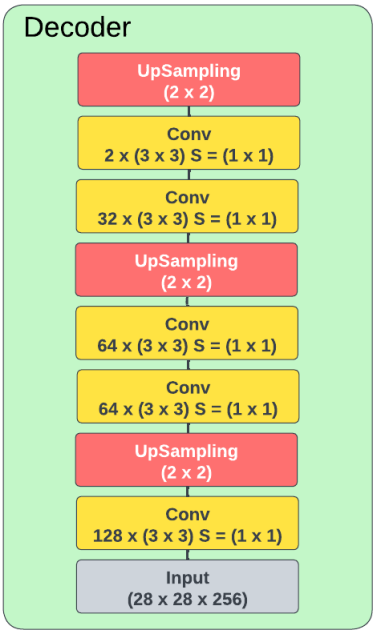


Figure 3: Fusing the Inception embedding with the output of the encoder

Fusion Layer: The fusion layer takes the feature vector from the Inception model and replicates it $HW/8^2$ times. The replicated feature vector is then concatenated to the feature volume outputted by the encoder along the depth axis (as shown in Fig. 3). This results in a single volume with the encoded image and the mid-level features of shape $H/8 \times W/8 \times 1256$.

By mirroring the feature vector and concatenating it several times, we ensure that the semantic information conveyed by the feature vector is uniformly distributed among all spatial regions of the image. Finally, we apply 256 convolutional kernels of size 1×1 with ReLU activation function to the resulting volume, ultimately generating a feature volume of dimension $H/8 \times W/8 \times 256$.

Decoder: The decoder takes the feature volume from the fusion layer and applies a series of convolutional and up-sampling layers to obtain a final layer with dimension $H \times W \times 2$.
Up-sampling is performed using the basic nearest neighbor approach, which doubles the height and width of the output compared to the input. Each convolutional layer uses a ReLU activation function, except for the last one, which uses a hyperbolic tangent function.



3 Experiments

3.1 Dataset

We chose the MIRFLICKR25000 dataset during our experiments because it is a large and diverse dataset of images. This makes it a valuable resource for training and evaluating image colorization algorithms. The dataset is also challenging because the images vary widely in terms of their content, lighting, and quality. This helps to ensure that the algorithms we developed are able to generalize to a wide range of images.

The MIRFLICKR25000 dataset is a collection of 25,000 images downloaded from the Flickr photo-sharing website. The images were selected to be a diverse set of natural scenes, including landscapes, cityscapes, and objects.

3.2 Training

We split the dataset of 25,000 images into 90% training data, 5% validation data, and 5% testing data.

The MSE loss is backpropagated through the network to update the model's parameters. We employed the Adam optimizer, enhanced with an exponential decay schedule to adjust the learning rate over time. This schedule incorporates an initial learning rate of 0.0001 and a decay rate of 0.9. Additionally, our training was conducted in batches, with each batch containing 64 samples.

We used the TensorFlow framework to implement and train the model.

3.3 Evaluation Metrics

3.3.1 PSNR

Peak Signal-to-Noise Ratio (PSNR) serves as a fundamental metric in our evaluation of image colorization results. PSNR quantifies the quality of a colorized image by measuring its similarity to the original grayscale version. Calculated through the mean squared error (MSE) between corresponding pixels of the original and colorized images, PSNR provides a numerical score that reflects the degree of pixel-wise fidelity. A higher PSNR value, usually expressed in decibels (dB), indicates a closer resemblance between the colorized and original images, with lower pixel-wise distortion. Given its simplicity, widespread acceptance, and ease of calculation, we opted for PSNR to assess the technical quality of our colorization outcomes which can be calculated using the following equation.

$$PSNR = 10 \log_{10}(\frac{R^2}{MSE}) \tag{1}$$

where R is the dynamic range of the image. While PSNR offers valuable insights into the pixel-level fidelity of colorized images, we acknowledge its limitations. It may not fully account for perceptually relevant distortions and tends to prioritize pixel-wise errors. As we delve deeper into our analysis, we also consider alternative metrics, such as Structural Similarity Index (SSIM) . Which takes into account perceptual factors and human visual perception, complementing the technical assessment provided by PSNR. Our comprehensive evaluation strategy aims to provide a well-rounded understanding of image colorization quality, taking both technical and perceptual aspects into consideration.

3.3.2 SSIM

In addition to Peak Signal-to-Noise Ratio (PSNR), we employed the Structural Similarity Index (SSIM) as a crucial metric in our image colorization evaluation. SSIM assesses image quality by comparing structural information and patterns between the original grayscale image and its colorized counterpart. Unlike PSNR, which focuses primarily on pixel-wise differences, SSIM takes into account luminance, contrast, and structure, making it well-suited for evaluating perceptual quality. A higher SSIM score signifies a more faithful colorization that maintains not only pixel fidelity but also structural integrity, making it a valuable complement to our assessment which is calculated using the following equation.

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2}$$

where c_1 and c_2 are small positive constants introduced to stabilize the division and adjust the sensitivity of SSIM to image structure and contrast. One of the key advantages of using SSIM is its sensitivity to perceptually relevant distortions and its ability to capture visual quality more comprehensively. By incorporating SSIM into our evaluation framework alongside PSNR, we gain a holistic perspective on the strengths and weaknesses of different colorization models and conditions. While PSNR highlights the technical quality of colorization, SSIM provides insights into how well colorized images preserve the visual patterns and structures present in the original grayscale images. This combined analysis allows us to make informed judgments about the overall image colorization quality, considering both technical and perceptual aspects, which are crucial for real-world applications where visual appeal and fidelity are essential.

3.4 Modifications To The Architecture

During our experimentation, we initially trained the baseline model, but encountered challenges with its convergence rate. Despite training for an extended duration, the model exhibited slow convergence and persistently high loss values. The output results also posed a significant concern, manifesting as predominantly grayish or brownish images. These limitations prompted us to explore modifications

to the model architecture and training process to address these issues and enhance the overall performance of our image colorization system.

3.4.1 Deeper Network

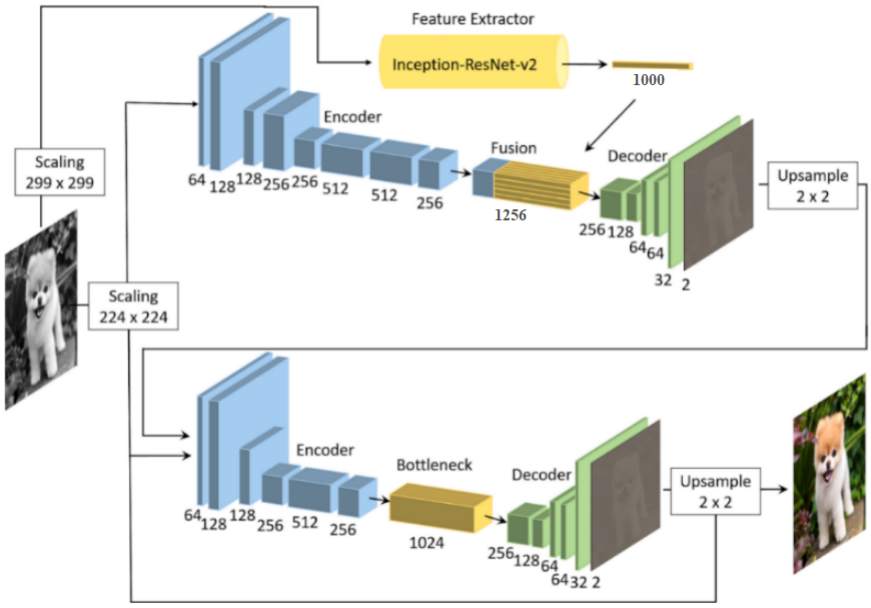


Figure 4: An overview of the deeper model architecture.

In an attempt to address the limitations of the baseline model, we tried a more ambitious approach by introducing a deeper network architecture. We concatenated the output of the baseline model with the L channel of the original image along the depth axis, forming a hybrid input. This hybrid input was then passed through an encoder-bottleneck-decoder network structure, which is similar to the baseline architecture. However, in this modified setup, the encoder component had fewer layers, and the bottleneck layer was defined as a convolutional layer with a substantial 1024 kernels, each sized 5×5.

However, despite these architectural enhancements, we observed that there was no significant improvement over the baseline model. In fact, the model exhibited even slower convergence, and the output images remained predominantly brownish or grayish in hue, indicating the need for further exploration and refinement in our approach to achieve the desired colorization results.

3.4.2 Baseline + Batch Normalization + Max Pooling

In our ongoing effort to improve the performance of the baseline model, we made further refinements to its architecture. We introduced a series of modifications that had a significant impact.

First, we incorporated batch normalization layers after every convolutional layer

and positioned them before the activation functions (as shown in Fig. 5). This adjustment played a pivotal role in the model's success.

Second, we changed our convolution layers. We replaced those with stride (2x2) with convolution layers with stride (1x1), which were followed by max pooling layers (as shown in Fig. 6).

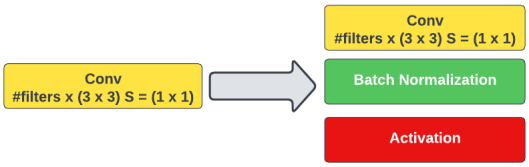


Figure 5: Adding a batch normalization layer between the convolution layer and its activation

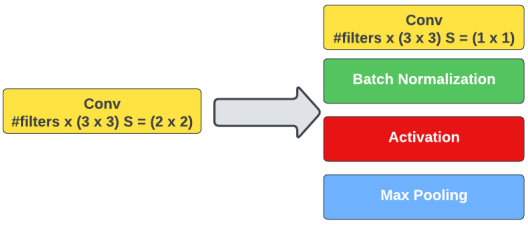
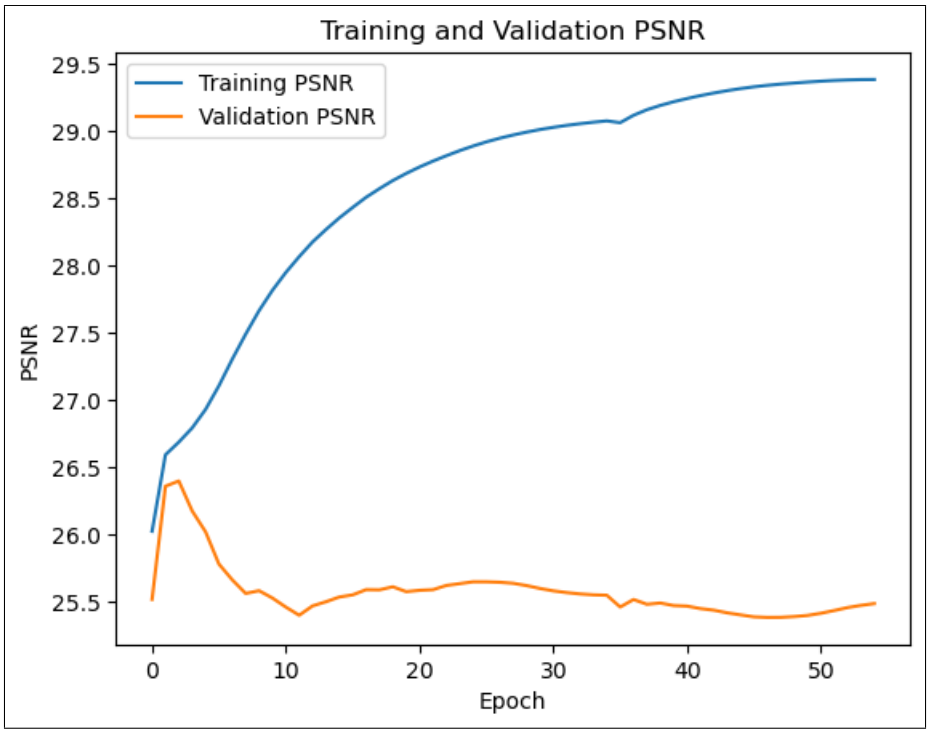


Figure 6: Replacing convolution layers with stride (2x2) with convolution layers with stride (1x1), followed by max pooling layer

This revamped model showed considerable promise. It exhibited robust convergence and effectively colored high-level image components such as the sky, sea, and forests. However, we recognize that there is room for improvement, particularly in the colorization of small details.

Another challenge we encountered was the onset of overfitting after a certain point in training. This can be addressed by using regularization techniques.

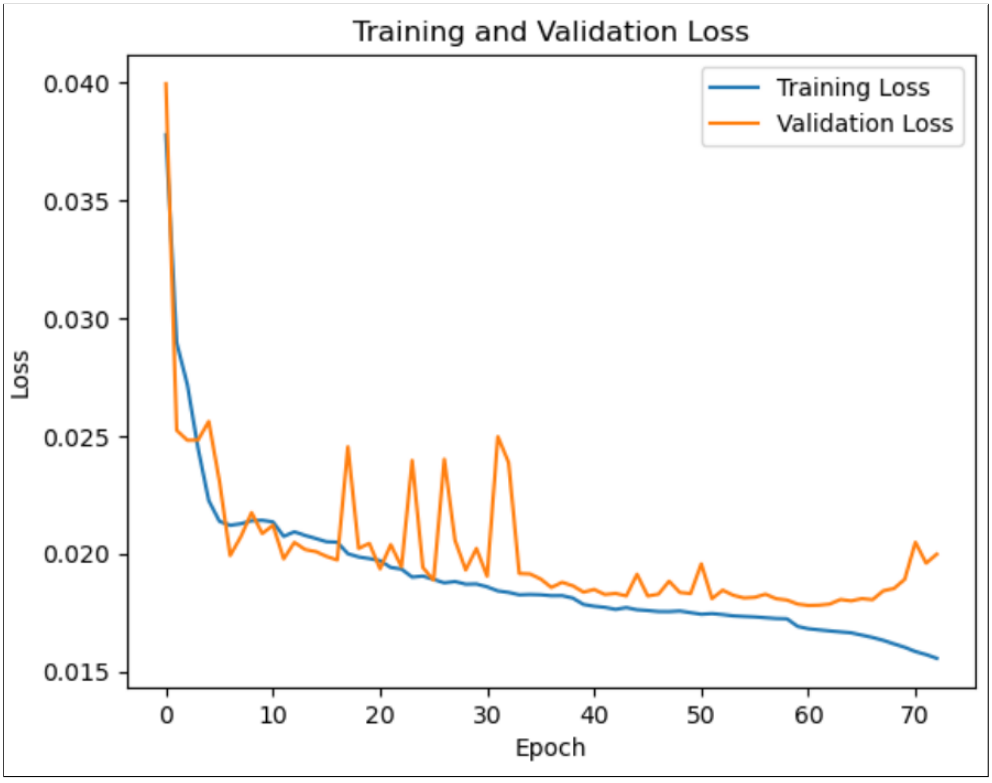
We monitored the training and validation performance using key metrics, as depicted in the plots below:

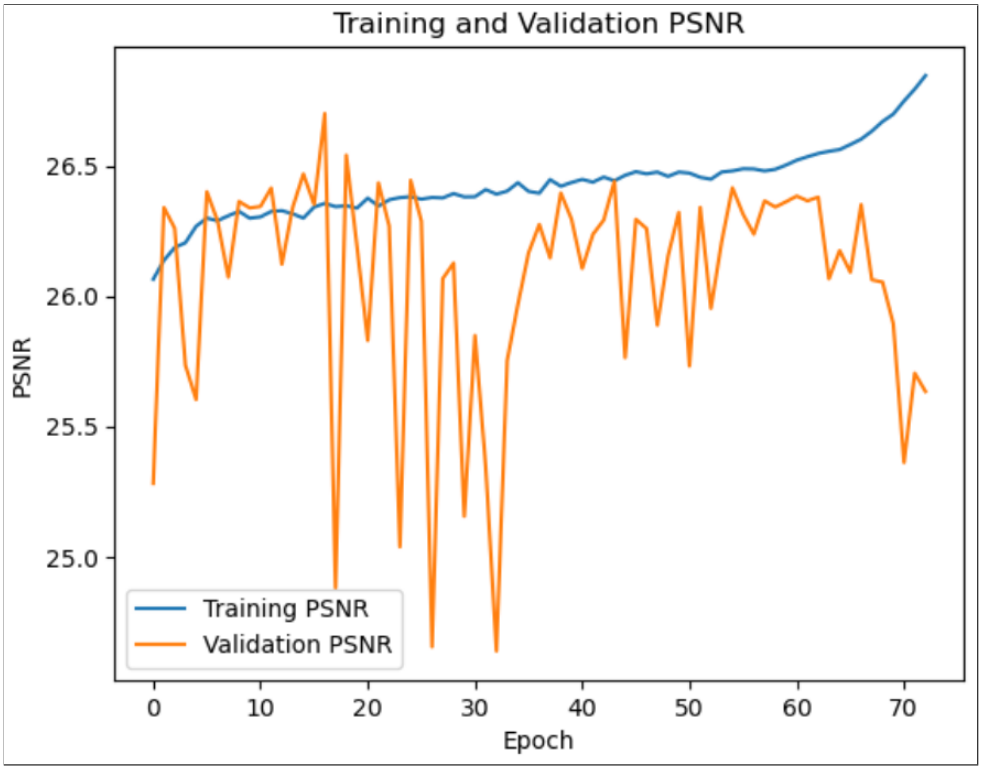


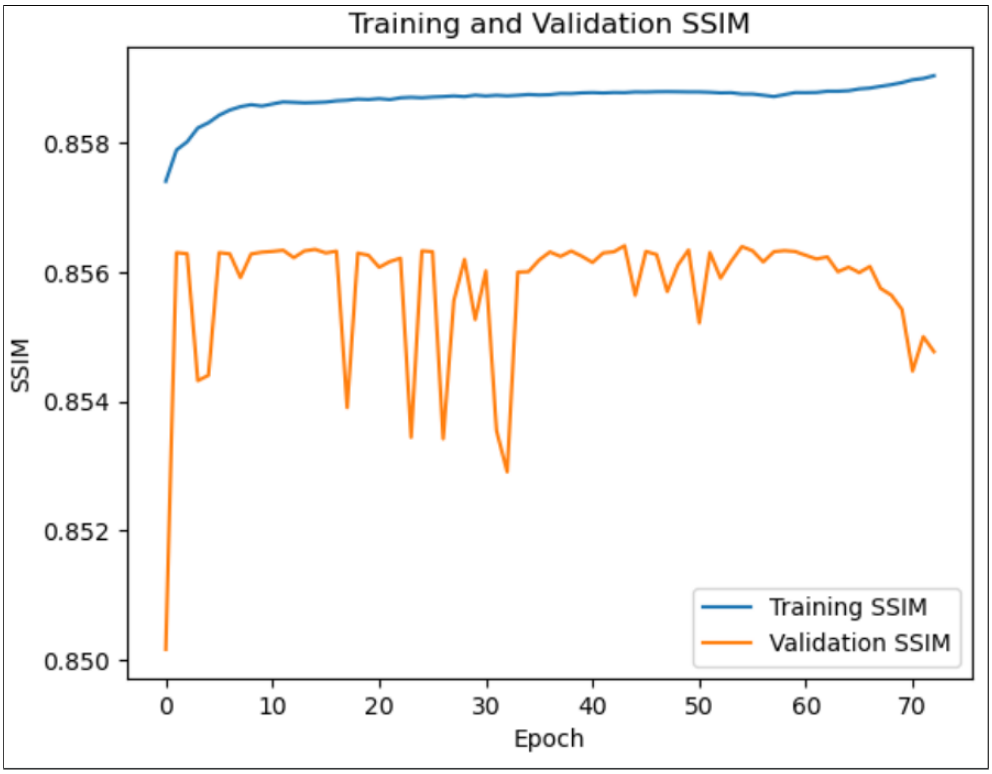
3.4.3 Baseline + Batch Normalization + Max Pooling + L2 Regularization

We explored the implementation of L2 regularization, with a regularization factor set to 0.01. However, this adjustment did not yield the desired results. Instead, we observed a regression in the colorization quality. The output images regressed

to being predominantly brownish and grayish once again. Additionally, this regularization strategy seemed to make the model increasingly sensitive to even minor variations in its parameters, as evidenced by the metrics tracking our progress, illustrated in the plots below.





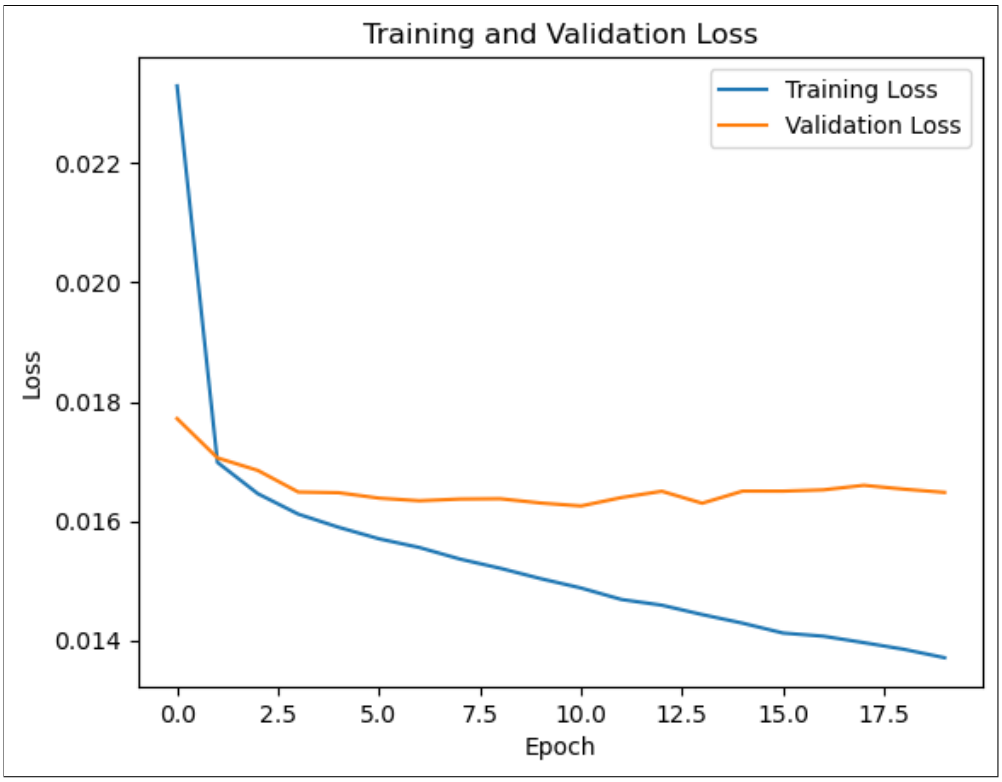


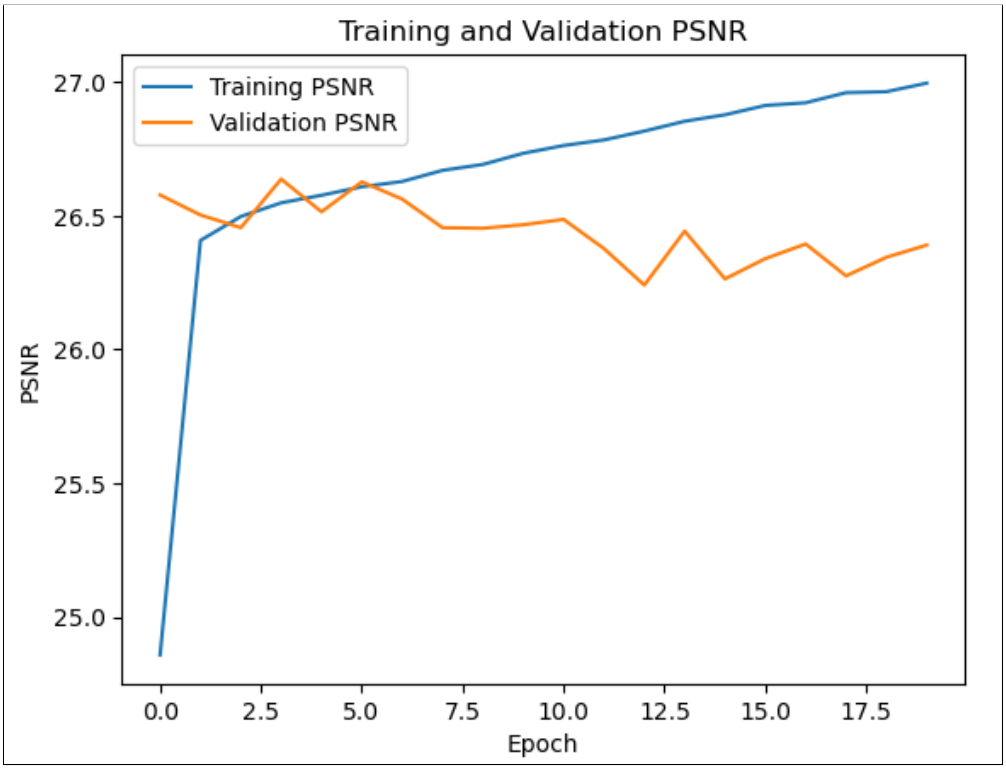
3.4.4 Baseline + Batch Normalization + Max Pooling + Spatial Dropout

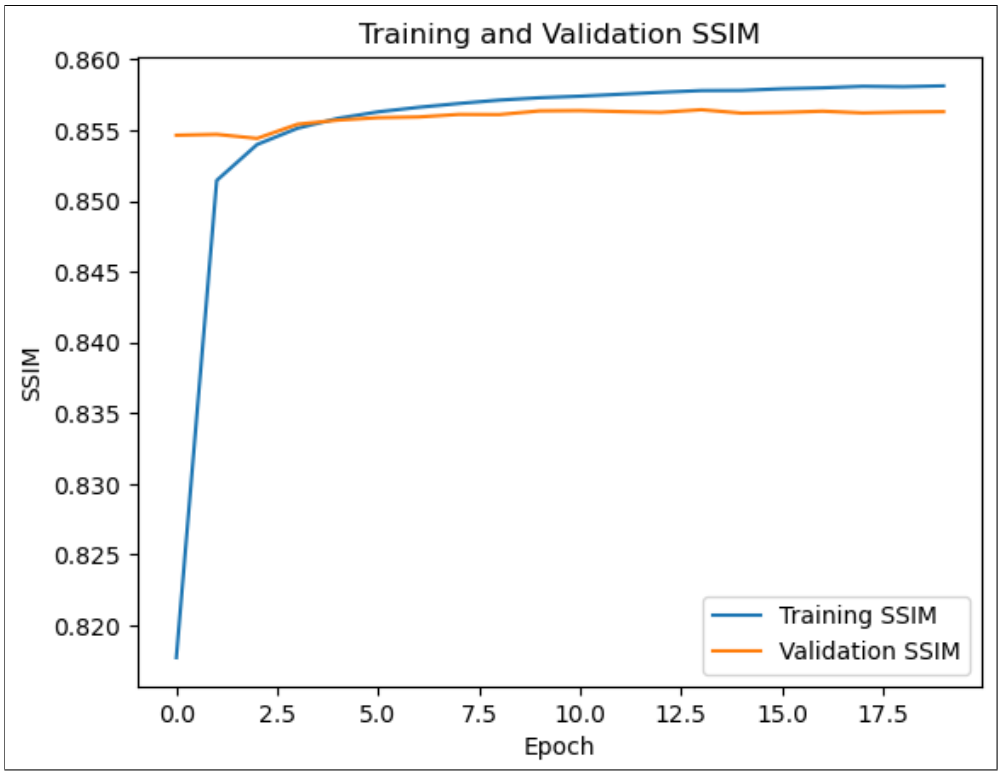
In an effort to mitigate overfitting , we introduced SpatialDropout2D layers with a dropout rate of 0.2 after every other convolution block in our architecture. Here, we define a convolution block as a sequence of operations consisting of convolution, batch normalization, activation, and, in some cases, max pooling (as shown in Fig. 5, 6).

SpatialDropout2D is a specialized variant of dropout tailored for 2D input data, such as images in a Convolutional Neural Network (CNN). Unlike traditional dropout, which randomly deactivates individual elements within a feature map, SpatialDropout2D randomly drops entire feature maps or channels.

The inclusion of SpatialDropout2D indeed proved effective in mitigating overfitting, and the model exhibited significant promise. However, due to time constraints, we were unable to extend the training for an extended period. As seen in the plotted metrics below:


Training and Validation Loss
Training Loss
Validation Loss
Loss
0.022
0.020
0.018
0.016
0.014
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Training and Validation PSNR
Training PSNR
Validation PSNR
PSNR
27.0
26.5
26.0
25.5
25.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch



3.5 Transfer Learning

3.5.1 VGG16 as pre-trained encoder

This model uses transfer learning to leverage the knowledge learned by the VGG16 model on the ImageNet dataset. The VGG16 model is a deep convolutional neural network that has been trained to classify images into 1000 different categories. The VGG16 model is a very powerful model, but it requires a large amount of data to train. By using transfer learning, we can leverage the knowledge learned by the VGG16 model without having to train our own model from scratch.

The VGG16 model expects a 3-channel input image, but our input image is a single-channel grayscale image. To convert the input image to a 3-channel image, we tried two approaches:

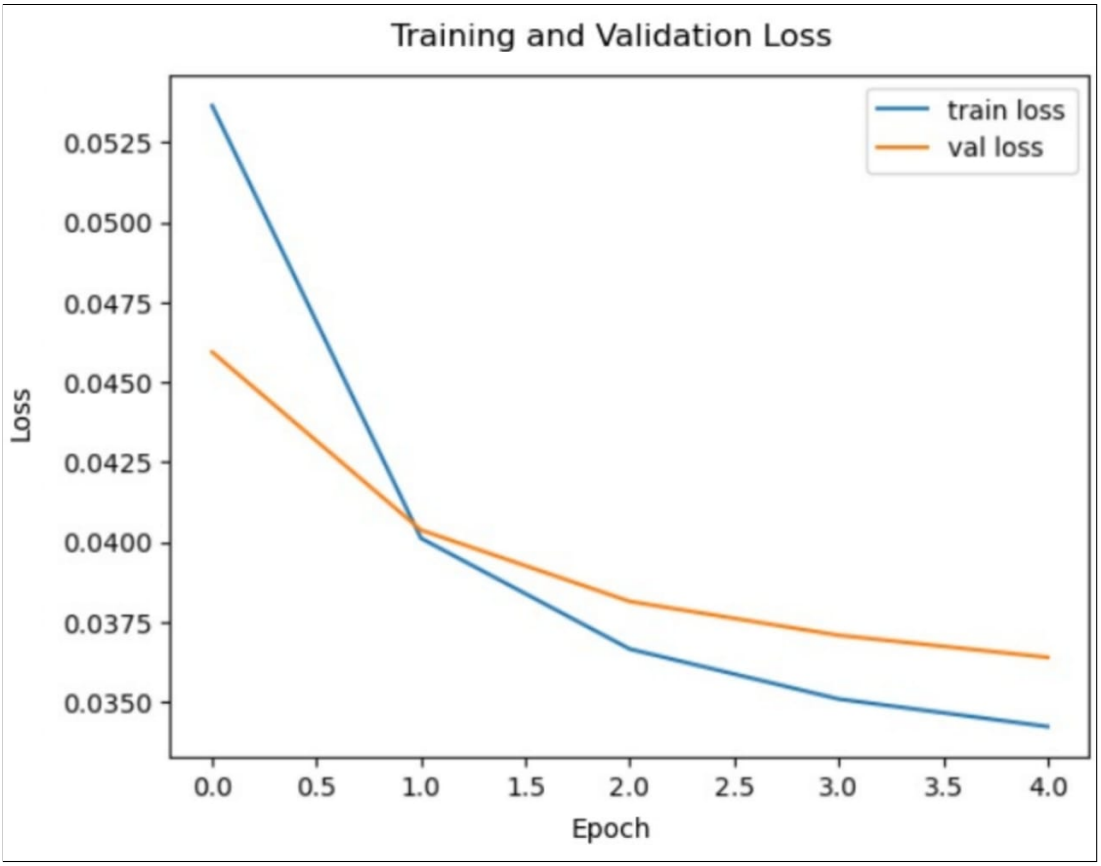
- Adding a new convolutional layer to the VGG16 model with 3 filters and a kernel size of (1, 1). This layer is used to replicate the single-channel input image to three channels.
- Repeating the single-channel input image three times and make input image is 3-channel image.

We found that the second approach is better.

Next, We set the trainable parameter to first layer is true but the remaining layers

are false to be fine-tuned during training to learn the specific task that we are trying to solve.

Finally, We make decoder take input with dimensions (7, 7, 512) with 5 blocks. First block is convolutional layer with 256 filters, each using a (3, 3) kernel and the ReLU activation function, while applying 'same' padding to maintain spatial dimensions. Following this, 4 block with 128 , 64, 32 and 16 filters. Finally, the model concludes with a final convolutional layer, featuring 2 filters, a (3, 3) kernel, and the hyperbolic tangent (tanh) activation function. This last layer's purpose is to predict the output image, and the output from this layer possesses dimensions of (224, 224, 2).



3.5.2 VGG19-based U-Net model

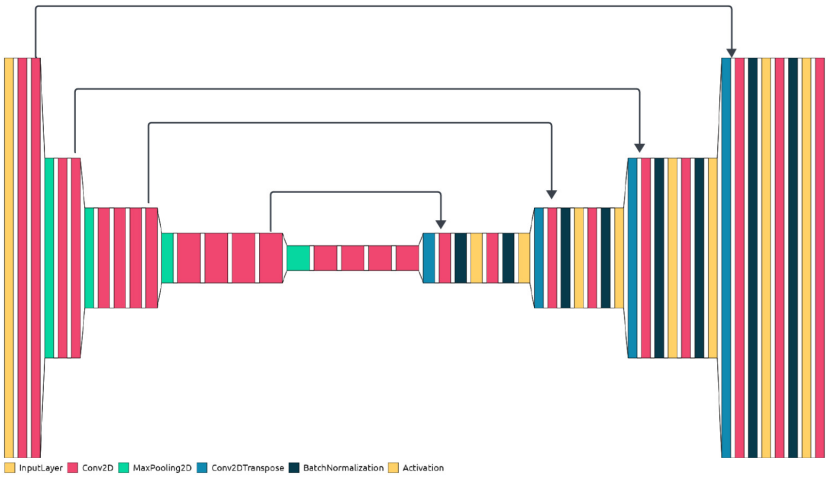


Figure 7: An overview of the VGG19-based U-Net model architecture.

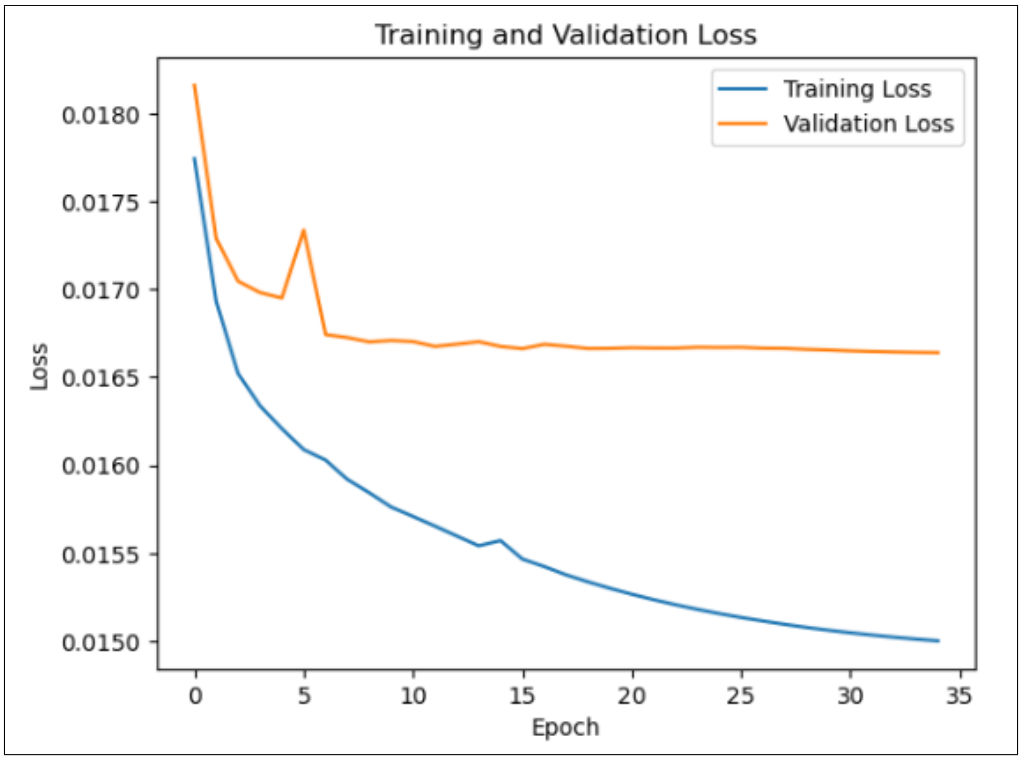
The VGG19-based U-Net model is a convolutional neural network (CNN). It is composed of three main components: the Encoder (based on VGG19 architecture), a Bridge layer, and a Decoder.

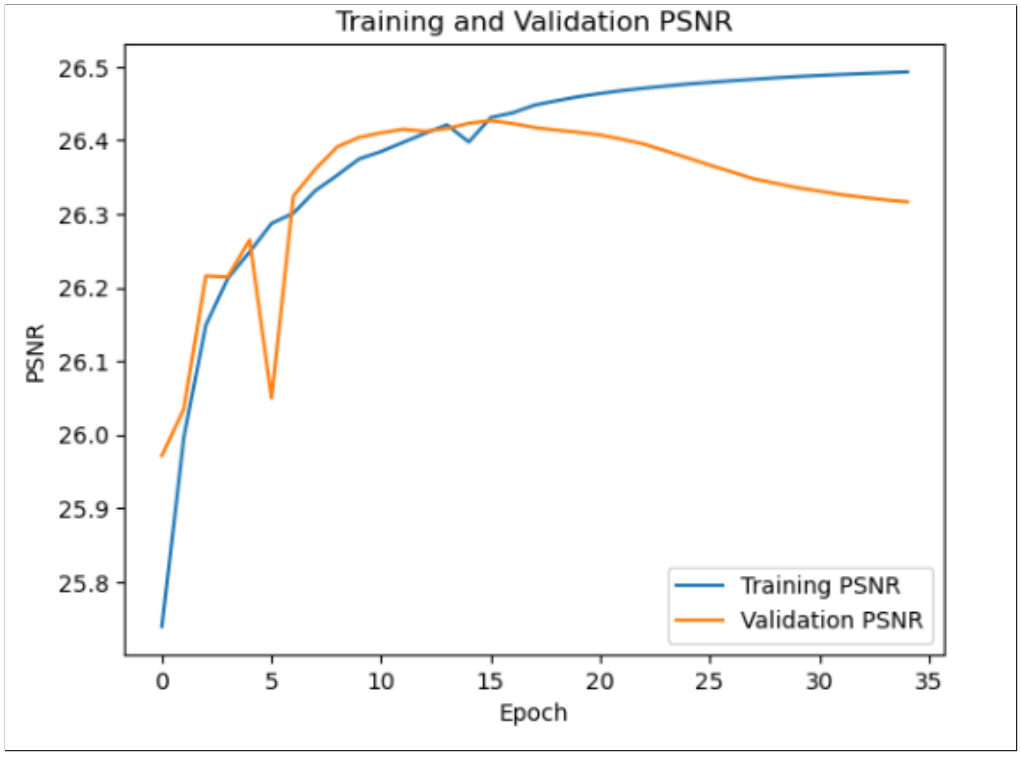
First, encoder utilizes the VGG19 architecture, a well-known CNN pre-trained on a large image dataset that take input image with 3-channel image, so we repeating the single-channel input image three times and make input image is 3-channel image.

Next, The bridge layer is a single convolutional layer that acts as a transition between the encoder and decoder, facilitating the flow of information.

Finally, The decoder consists of four blocks, each responsible for upsampling and feature extraction. First Decoder Block: Transposed convolutional layer with (2, 2) strides for upsampling. It concatenates encoder features and applies (3, 3) filter convolutions with ReLU activation. Following this, second, third and fourth decoder block that are similar to the first block, it handles upsampling, feature concatenation, and (3, 3) filter convolutions with ReLU activation.

The model concludes with a final convolutional layer, featuring 2 filters, a (1, 1) kernel, and the hyperbolic tangent (tanh) activation function. This last layer's purpose is to predict the output image, and the output from this layer possesses dimensions of (224, 224, 2).


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch
0.0180
0.0175
0.0170
0.0165
0.0160
0.0155
0.0150
0
5
10
15
20
25
30
35


Training and Validation PSNR
Training PSNR
Validation PSNR
PSNR
Epoch
26.5
26.4
26.3
26.2
26.1
26.0
25.9
25.8
0
5
10
15
20
25
30
35

4 Discussion

4.1 Results

Upon completing the training and fine-tuning phases, we subjected the testing data to our refined model, namely the Baseline + Batch Normalization + Max Pooling configuration, which demonstrated the most promising results in our experiments. The results were promising, with some images being colorized very well, closely resembling photorealistic representations.

However, it is important to note that the model's performance was limited by the relatively small size of the training dataset. As a result, the network was better at colorizing certain image features than others. Natural elements such as the sea or vegetation were often well-recognized and colorized accurately. However, the model's performance varied when faced with specific objects or intricate details.

Another observation was that some colorized images were under-saturated, often producing a grayish or neutral color. We believe that this is because the network, in its attempt to minimize the loss between the images, tends to make conservative predictions. When the network is uncertain about the appropriate colorization, it defaults to assigning a neutral gray color.

Figure 8: Good results

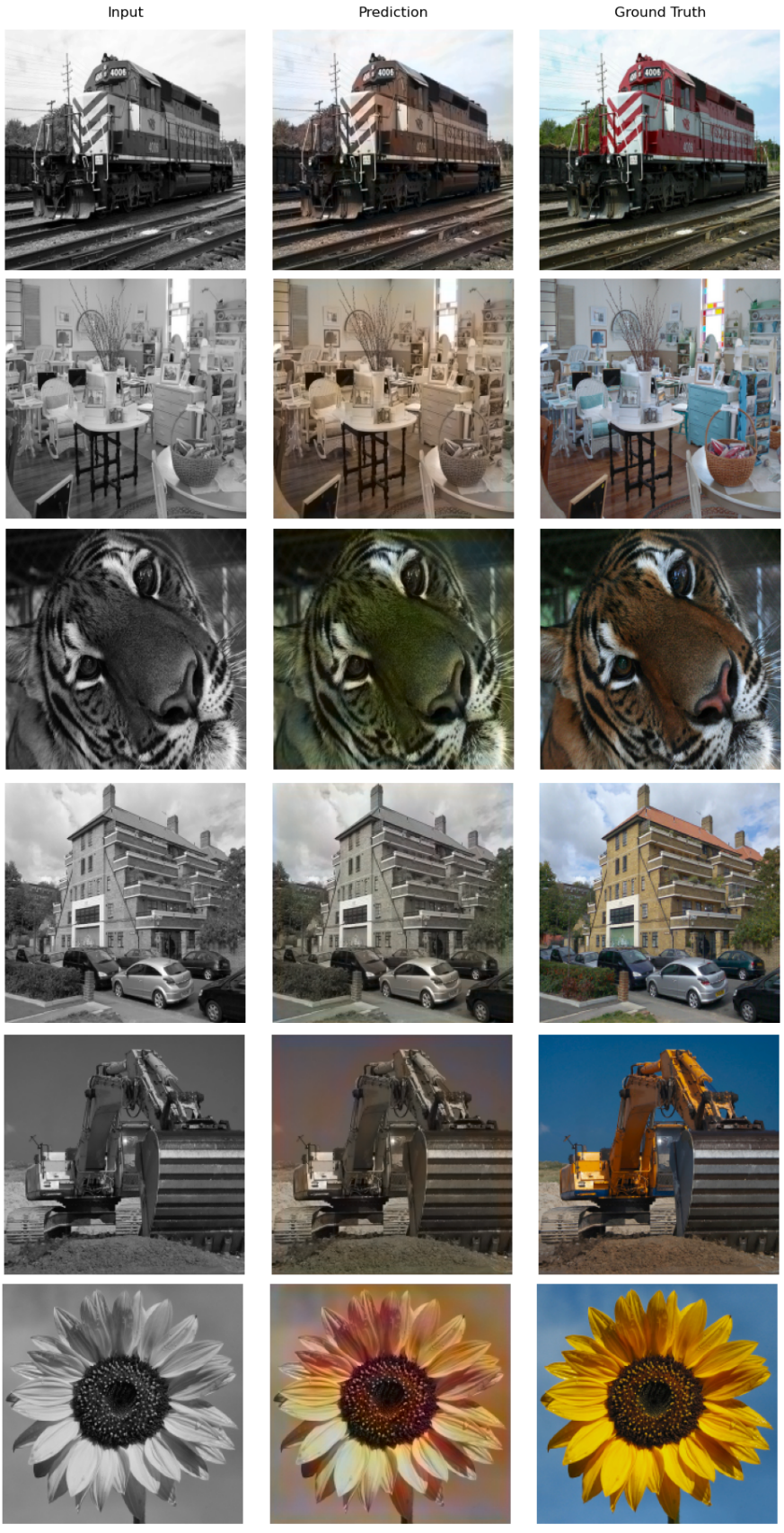


Figure 9: Poor results

4.2 Future Work

Overall, the results of our experiments are promising. Yet, there is still room for improvement, especially in terms of the model's ability to colorize specific objects and intricate details. Regrettably, due to time constraints, we were unable to delve deeper into the exploration of alternative regularization techniques and further model refinements. Looking ahead, we believe that with a larger training dataset and further refinements to the model, our model can achieve even more impressive results.

References

[1] Baldassarre Fe, Diego González Morín, and Lucas Rodés-Guirao. "Deep Koalarization: Image Colorization using CNNs and Inception-Resnet-v2." arXiv preprint arXiv:1712.03400 (2017).