Colorization of black and white images

BOISSOT Aurélien
aurelienandremanueljean.boissot@studenti.unipd.it

MOULON Florent
florentgillesgerard.moulon@studenti.unipd.it

SKOWRONEK Arthur
arthur.skowronek@studenti.unipd.it

Abstract

Image colorization is a complex task due to the ambiguity of color prediction. We propose a deep learning approach using a U-Net-inspired convolutional neural network with dilated convolutions and without pooling layers to retain spatial details. Our model is trained on a dataset of 3,901 images of fruits and vegetables, ensuring diversity in object appearance and backgrounds.

To address color imbalance, we apply a weighted cross-entropy loss, enhancing rare color representation. We also use annealed-mean decoding with temperature scaling to improve color vibrancy. Comparing different loss functions, we find cross-entropy yields the most realistic results.

A Turing test shows that 70% of real and 43% of generated images are perceived as realistic, demonstrating our model's effectiveness. Efficient with low-resolution (100×100) images, our approach requires no user input, learning colorization from context. Code and dataset are publicly available on GitHub.

1. Introduction

Old photos or movies are often in black and white. Automating the colorization process can save significant time and provide additional benefits, such as reducing storage space or communication bandwidth by storing only black-and-white images and reconstructing color when needed.
Formally, if we decompose color into three variables using the CIE Lab color space (L, a, b), the task involves predicting the a and b channels based on the luminance (L).
Problem Formulation: The goal of image colorization is to predict the missing chrominance information for a grayscale image, typically generated through a function $\Phi(\cdot)$ that converts an RGB image to grayscale, such as:

$$I_g = 0.2989 \times I_r + 0.5870 \times I_g + 0.1140 \times I_b$$

Rather than restoring all three RGB channels, the task is simplified by working in alternative color spaces like CIE Lab, where only the chrominance channels (a and b) need to be predicted, given the luminance (L) as input.
Challenges: This problem is highly challenging due to factors such as variations in illumination, light and shadow interactions, and occlusions. To achieve successful colorization, the algorithm can leverage contextual information from the image. For instance, skies are typically blue, grass is often green, and the sun appears yellow. However, ambiguity arises with objects like a T-shirt or an apple, which can exist in multiple realistic colors. This makes it crucial not to aim for exact color replication but rather to generate a plausible image that appears realistic to a human observer.
Our results: In our approach, we did not incorporate a perceptual realism metric into the loss function, which primarily compares differences between the generated output and the ground truth. However, to evaluate the quality of our final results, we conducted a Turing Test. We designed a form that randomly mixed 20 ground truth images with 20 generated images.[1] Participants were asked to identify whether the images were real or generated. From this evaluation, 70% of ground truth images and 43% of generated images were considered realistic by human evaluators.

2. Related work

2.1. Other works

Image colorization has undergone significant advancements, particularly with the advent of deep learning. Traditional methods relied on manual annotations or basic machine learning techniques, but recent approaches leverage convolutional neural networks (CNNs) and generative adversarial networks (GANs) to automate and enhance the process.

One of the earliest CNN-based methods, Deep Colorization [5], utilized convolutional layers alongside joint bilateral filtering to improve image quality but faced limitations in learning efficiency. Later, Colorful Image

[1] https://florentmoulon.github.io/Image-Colorization-Turing-Test/

Colorization [15] introduced a novel rebalancing loss to handle class imbalance, improving color diversity. Methods like U-Net-based colorization [9] further refined the process by incorporating skip connections for better spatial consistency. However, these models struggled with ambiguous regions, such as objects with multiple plausible colors.

User-guided approaches, such as Scribbler [12] and Real-Time User-Guided Colorization [1], allowed users to specify colors for specific regions, combining manual input with automated processing. Although effective for specific applications, these methods required significant user involvement, limiting scalability for large datasets.

Domain-specific techniques like SAR-GAN [14] and Infrared Colorization [11] tailored algorithms for specific modalities, achieving superior results within their domains. However, these approaches lacked generalizability to broader datasets. Similarly, exemplar-based models, such as Deep Exemplar-Based Colorization [8], utilized reference images to transfer colors, offering diverse outputs but heavily relying on the quality and relevance of exemplars.

Recent innovations focus on diverse and multi-modal colorization. Diverse models like ChromaGAN [13] and CycleGAN-based approaches [10] generate multiple plausible outputs, enhancing aesthetics. Multi-modal networks, such as Text2Colors [3] and Depth-Aware Colorization [6], incorporate additional inputs like text or depth. While expanding applications, these methods demand complex training data and higher computational resources.

2.2. Our approach :

Compared to prior methods, our approach emphasizes generating realistic and perceptually convincing outputs without the need for direct user input or domain-specific data. We adopt a smaller and more basic network architecture, optimized to work efficiently within limited computational resources. Our model is inspired by the "Colorful Image Colorization" method proposed by Richard Zhang [15], leveraging its robust design while tailoring it to our specific constraints. We train our model exclusively on images of fruits and vegetables, using a resolution of 100 × 100 pixels. This allows us to focus on a manageable dataset and resolution, balancing simplicity and effectiveness while ensuring practical applicability in resource-constrained environments.

3. Dataset

3.1. Dataset Overview

For our image colorization task, we opted to work with images of fruits and vegetables, each resized to 100x100 pixels.
Given our computational constraints, we chose to specialize our model on a specific category of images. We choose to focus on Fruits and vegetables for several reasons.
First, these objects can be distinctly categorized into classes, ensuring a balanced representation across different types. This facilitated dataset organization and helped mitigate class imbalance issues. Second, fruit and vegetable images are widely available, making dataset acquisition more convenient. Finally, their natural color variations (e.g., apples ranging from red to green) allowed the model to generate plausible predictions without requiring exact replication of the ground truth colors.
The chosen resolution maintain a balance between computational efficiency and image clarity, ensuring that both the model and human evaluators could easily interpret the images.
Our final dataset consists of 3,901 images distributed across 24 classes, with each class containing between 100 and 200 images. We kept 5 images of each class for the test dataset and then split the remaining images into validation and training sets with a proportion of 0.8.

3.2. Dataset Collection

Initially, we used the Natural Color Dataset[8] proposed by Anwar et al [2], specifically designed for colorization tasks. However, with only 723 images, this dataset was insufficient. To supplement it, we searched for additional fruit and vegetable image datasets available online.

While large datasets such as the Vegetable Image Dataset[6] were considered, they were overly homogeneous. Instead, we combined multiple smaller datasets from sources like Kaggle, ensuring diversity in image quality, backgrounds, and photographic conditions. This approach yielded a dataset that includes high-quality studio images, amateur photographs, and in-context representations.

The final dataset was compiled using 6 different sources publicly available that you can find in annex 7.1.

3.3. Preprocessing

3.3.1 Filtering

Since some datasets were originally designed for classification tasks, they contained non-photographic images such as illustrations, logos, and drawings. Additionally, certain datasets included cooked or processed food items that no longer resembled their raw form. We manually filtered these images, ensuring that only real photographs of fresh

fruits and vegetables were retained.

3.3.2 Class Rebalancing

Our initial dataset contained over 5,200 images spanning 34 classes. To prevent class imbalance, which could bias the model, we imposed constraints on class sizes. Specifically, we retained only classes with at least 100 samples and limited the maximum number of images per class to 200 by selecting the first 200 instances.

3.3.3 Formatting

The collected images varied in resolution and aspect ratio. To standardize them to our 100x100 pixel format, we applied padding instead of cropping. White borders were added to rectangular images to maintain the full content, avoiding the removal of key visual features. This approach was particularly suitable as many images already had white backgrounds, ensuring consistency in appearance and preventing unintended distortions.

4. Method

Our approach uses a deep CNN with an encoder-decoder architecture. This CNN is trained to map a grayscale input image to a probabilty distribution over 313 possible color value outputs. We will study the architecture of our model.

4.1. Network architecture

We use a U-net like architecture, using dilated convolution and without pooling layers.

Encoder: Our model uses ConvBlocks to extract features. Each ConvBlock contains two convolutional layers with batch normalization, ReLU activation, and progressive dropout (the rates vary from 0.05 for the first ConvBlock, to 0.3 for the deepest ConvBlocks).
We use dilated convolutions (dilation 2 and 4) in deeper layers which expand the receptive field exponentially. This allows the model to capture the global context without downsampling.
Decoder: The decoder uses transposed convolutions for upsampling (with batch normalization, ReLU activation and dropout), followed by ConvBlocks with decreasing dropout (from 0.2 to 0.05). The final convolution layer outputs a **313-channel probability map over quantized ab bins**. This layer is followed by a softmax.

4.2. Loss function

Weights initialization
In general, natural images have a skewed color distribution. Indeed, many images may have desaturated backgrounds, or lack vivid colors. This may affect the ability of our model to learn all colors. In order to address this issue, we rebalance the loss function using weights derived from the empirical color distribution.

The empirical color distribution represent the frequency of each of the 313 color bins q in the training data. We smooth this color distribution by mixing it with a uniform distribution. This ensures that no bin has a color probability of zero. We use an hyperparameter λ set to $\lambda = 0.5$ in order to balance the empirical color distribution with the uniform color distribution :

$$P_{smooth,q} = (1 - \lambda) \cdot P_{emp,q} + \frac{\lambda}{Q}$$

We then calculate weights inversely proportionally to the smoothed probabilities to upweight rare colors. Finally, we normalize the weights to ensure that the expected weight value accross the dataset is 1.
This allows us to ensure the gradients for all colors are on a similar scale. Otherwise, the model may focus excessively on rare colors, and perform poorly on frequent colors.

The computation of these weights [15] allows us to mitigate class imbalance, by assigning higher weights to rare colors, and preventing overfitting to common colors.

Loss function

We now use these weights to implement a weighted cross-entropy loss function :

$$L_{\text{CE}} = -\sum_{q} w_q y_q \log(\hat{y}_q)$$

- where w_q is the normalized weight for the class q.

4.3. Annealed-mean decoding

The model outputs a probability distribution over 313 ab bins. To convert this discrete probability distribution into continuous values in the Lab color space, we use the annealed-mean decoding method. [15]
This process consists of two steps: we first apply temperature scaling on the probability, then we convert the probability distribution into a continuous representation by applying mean decoding.

Temperature scaling
If we apply mean decoding right away, we may get images with desaturated colors. To alleviate this problem, we apply temperature scaling which aims to adjust the sharpness of the probability distribution before decoding it into a continuous color value. To do so, we apply the following formula :

$$P_{T,q} = \text{softmax}(\frac{\log P_q}{T})$$

where P_q is the original probability distribution predicted by the model, and T is the temperature parameter that controls the distribution sharpness. If $T = 1$, the distribution is left unchanged, while a lower temperature increases the sharpness of the color distribution, but may decrease the spatial coherence of the result. As stated by Zhang et al. [15], a temperature $T = 0.38$ provides a good balance between obtaining vibrant colors, while keeping the spatial coherence of the result.
By applying temperature scaling, we reweights the probabilities to get more vibrant colors, and further suppress low-probability bins.

Mean decoding
Using the adjusted probability distribution $P_{T,q}$ computed above, we can compute the final continuous ab values using the mean decoding formula:

$$ab = \sum_{q=1}^{313} P_{T,q} \cdot \mu_q$$

- where μ_q is the centroid of each of the 313 color bins.

This final step converts the discrete probability distribution into a continuous color representation.

4.4. Training strategy

Our learning strategy is designed to prevent overfitting while learning efficiently. We will detail the main components of our learning strategy.

Optimizer
We use the Adam optimizer with a learning rate of $\eta = 10^{-4}$ to ensure stable updates and a weight decay of $\lambda = 10^{-4}$ to apply L2 regularization, penalizing large weights and reducing overfitting.
Early stopping
We implemented an early stopping mechanism to prevent overfitting, by stopping the training when the performance of the model stops improving on the validation set. We added a patience parameter $p = 4$ so that the training continues for 4 epochs after it stops improving to try to escape a local minima. We also use a minimum delta $\delta = 0.001$ so that a progress below this threshold is not considered as progress
Progressive dropout
We use increasing dropout in the encoder, and decreasing dropout in the decoder. This ensures that the model does not rely too heavily on some specific neurons, making it more robust to variations, and preventing overfitting in the deeper layers.

5. Experiments

To achieve the best performance in image colorization, we experimented with various initialization strategies for bin probabilities.

5.1. Bin initialization

5.1.1 Uniform Probability Initialization

Initially, we set all bin probabilities to 1, meaning that every color was treated equally without any prior knowledge of its occurrence in the dataset.
Result: The model struggled to learn meaningful color distributions, resulting in slow training, significant overfitting, and a lack of diversity in color representation in the generated images.

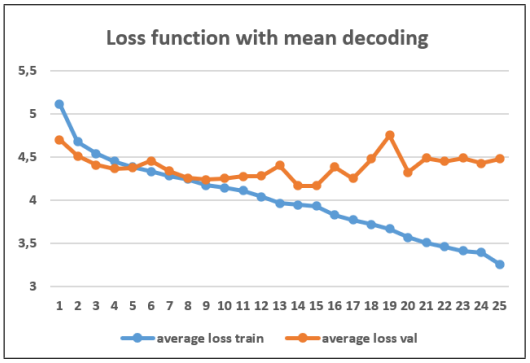

Figure 1. Loss function with mean decoding

5.1.2 Data-Driven Probability Initialization

Next, we implemented an algorithm to compute the color distribution directly from the training dataset. Each bin's probability was initialized based on its empirical frequency in the dataset.
Result: The model predominantly produced brownish images, with rare colors being underrepresented, resulting in a lack of vibrancy.

Figure 2. Brownish images

5.1.3 Class-Balanced Reweighting (Paper Method)

Finally, we applied the approach described in the reference paper [15], which addresses class imbalance by reweighting the loss of each pixel according to the rarity of its color as described in the method section.
Result: The model successfully captured a more balanced color distribution, resulting in generated images that appeared more natural with better color diversity and fidelity.

5.2. Loss function

As discussed in the method section, we ultimately decided to implement the cross-entropy loss function. This choice was motivated by its widespread use as a standard loss function, notably in Zhang et al. [15]. Additionally, our experiments demonstrated that it produced the most realistic generated images among the three loss functions we tested.

Beyond cross-entropy, we also implemented the **gradient loss**, which is computed by comparing the horizontal and vertical gradients of the predicted and ground-truth images:

$$L_{\text{grad}} = \sum ||\nabla_h \hat{y} - \nabla_h y||_2^2 + ||\nabla_v \hat{y} - \nabla_v y||_2^2$$

where: In this context, ∇_h and ∇_v represent the horizontal and vertical gradients, respectively, while $\hat{y}$ and y denote the predicted and ground-truth images.
We chose to experiment with gradient loss because it was employed in the work of Deshpande et al. [7] and discussed in the survey by Anwar et al. [2]. The intuition behind this loss function is that it encourages spatial smoothness and preserves structural details in the generated images. However, our results indicated that it performed poorly, leading to unrealistic colorization, and we quickly decided to abandon it.

The third loss function we tested was **focal loss**, a variant of cross-entropy loss designed to handle class imbalance:

$$L_{\text{foc}} = -\alpha(1 - p_t)^\gamma \log(p_t)$$

where:

- $p_t = \sum y_q \hat{y}_q$ is the predicted probability for the true class
- α is a weighting factor (set to 0.25 in our case) to balance class importance
- γ is a focusing parameter (set to 2 in our case) to down-weight easy examples

To evaluate the effectiveness of these loss functions, we conducted a comparative analysis using a lightweight **Turing test**. Specifically, we counted how many generated images appeared realistic and averaged the results. While this method is inherently subjective and may introduce bias, it provided a practical approach given our limited resources.

The results of our evaluation are summarized in the following figures:

Number of categories with level of performance			
% perceived as real	**Loss 1**	**Loss 2**	**Loss 3**
0%→25%	13	24	14
25%→50%	5	0	7
50%→75%	6	0	3
75%→100%	0	0	0

Table 1. Comparison of the 3 loss functions performance by class in a lightweight Turing test

	Loss 1	**Loss 2**	**Loss 3**
% perceived as real	27%	0%	20%

Table 2. Comparison of the average performance of the 3 loss functions

Based on these findings, we concluded that the cross-entropy loss function yielded the best results in terms of realism and overall performance. Consequently, we adopted it as the primary loss function for our final model.

5.3. Dropout

Our final experiment focused on dropout regularization, as we observed early overfitting in our model.
As illustrated in the following graph: The loss value on the

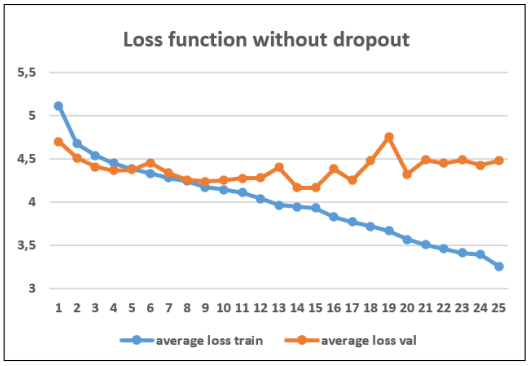

Figure 3. Loss function without dropout

test dataset stops decreasing after approximately 10 epochs, showing a lot of fluctuations.
To limit this overfitting, we implemented dropout in our model. The dropout probability started at 0.1 for the first layer, increased incrementally by 0.1 up to 0.5 by the fifth layer, and then gradually decreased to 0.2 in the final layers (cf. Annex 7.3).

This technique significantly helped in preventing overfitting, but it was overly restrictive, causing our model to improve at a much slower rate. To optimize model perfor-

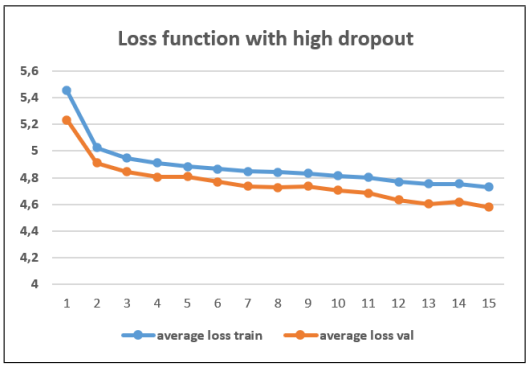

Figure 4. Loss function with high dropout

mance, we decided to reduce dropout rates, approximately halving the previous values (cf. Annex 7.4).

The resulting performance, depicted in the following graph, shows a significant improvement: = As seen in the

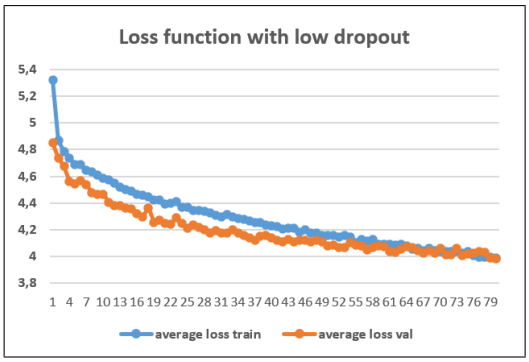

Figure 5. Loss function with low dropout

graph, this adjustment effectively limits overfitting while allowing the model to learn more efficiently. To compare these three models, we conducted a final, simplified Turing test, which clearly demonstrated the superiority of our last configuration. Hence, our final model is the one with a low

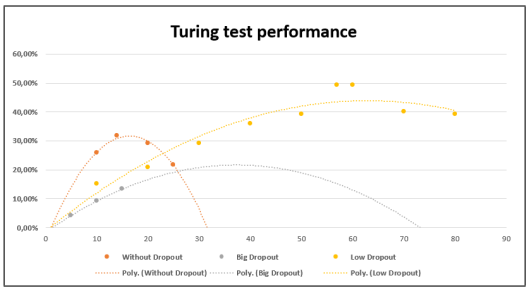

Figure 6. Turing Test performance

dropout rate and trained for 60 epochs.

5.4. Turing Test

To evaluate the performance of our final model, we conducted a Turing Test. Participants were shown 40 images, each displayed for 3 seconds, and asked to determine whether the image was an original picture or a recolorized version. From this evaluation, completed by 30 participants, 70% of ground truth images and 43% of generated images were perceived as real by human evaluators.
Based on these results, we computed the confidence interval for the proportion of recolorized images misclassified as real. The calculated confidence interval is [0.3662, 0.4855] with a confidence level of 95%.

Additionally, we tested the null hypothesis that there is no significant difference between the classification rates of real and recolorized images [4]. Using an independent t-test, we obtained a test statistic of 8.7308 and a p-value of 0.0000. This result allows us to reject the null hypothesis, indicating a statistically significant difference in perception between real and recolorized images. Although our model did not achieve an indistinguishable quality compared to ground truth images, it will still succeeded in deceiving participants in 36% to 48% of cases, demonstrating its ability to generate realistic colorizations.

6. Conclusion

With 43% of generated images being perceived as realistic (compared to 70% for ground truth images), we successfully developed a reasonably effective model given our limited computational resources. While not yet practical for real-world applications, the results demonstrate the potential of our approach within these constraints.

With more computational power, we could have explored additional hyperparameter tuning, such as adjusting the temperature parameter, which we left unchanged due to time constraints. Furthermore, our approach was based on a relatively older method from nine years ago; implementing more recent advancements, such as GANs or transformers, could have significantly improved performance. Expanding the size and diversity of our training dataset would also have helped enhance the model's ability to generate more realistic images.

If we were to continue this project, we could refine our architecture by introducing skip connections between layers, similar to ResNet, to improve gradient flow and model stability. Additionally, implementing k-fold cross-validation would allow for a more robust evaluation of model performance and better generalization. These improvements, combined with optimized training strategies, could help further bridge the gap between our model and real-world applications.

References

[1] Real-time user-guided image colorization with learned deep priors. *ACM Transactions on Graphics*.

[2] Saeed Anwar et al. Image colorization: A survey and dataset. *Information Fusion*, 114:102720, 2025.

[3] Hyojin Bahng et al. Coloring with words: Guiding image colorization through text-based palette generation. *arXiv*, 1804.04128, 2018.

[4] Yun Cao et al. Unsupervised diverse colorization via generative adversarial networks. In Michelangelo Ceci et al., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 151–166. Springer International Publishing, 2017.

[5] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. Deep colorization. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 415–423, 2015.

[6] Wei-Ta Chu and Yu-Ting Hsu. Depth-aware image colorization network. In *Proceedings of the 2018 Workshop on Understanding Subjective Attributes of Data, with the Focus on Evoked Emotions*, pages 17–23. Association for Computing Machinery, 2018.

[7] Aditya Deshpande et al. Learning diverse image colorization. *arXiv*, 1612.01958, 2017.

[8] Mingming He et al. Deep exemplar-based colorization. *ACM Transactions on Graphics*, 37(4):47:1–47:16, 2018.

[9] Zhengbing Hu et al. Grayscale image colorization method based on u-net network. *International Journal of Image, Graphics and Signal Processing*, 16(2):70, 2025.

[10] Bin Li et al. Image colorization using cyclegan with semantic and spatial rationality. *Multimedia Tools and Applications*, 82(14):21641–21655, 2023.

[11] Matthias Limmer and Hendrik P. A. Lensch. Infrared colorization using deep convolutional neural networks. *arXiv*, 1604.02245, 2016.

[12] Patsorn Sangkloy et al. Scribbler: Controlling deep image synthesis with sketch and color. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6836–6845, 2017.

[13] Patricia Vitoria et al. Chromagan: Adversarial picture colorization with semantic class distribution. *arXiv*, 1907.09837, 2020.

[14] Puyang Wang and Vishal M. Patel. Generating high quality visible images from sar images using cnns. *arXiv*, 1802.10036, 2018.

[15] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. *arXiv*, 1603.08511, 2016.

7. Annexes

7.1. Dataset sources

- Raghav R Potdar - Fresh and Stale Images of Fruits and Vegetables (https://www.kaggle.com/datasets/raghavrpotdar/fresh-and-stale-images-of-fruits-and-vegetables)
- Kritik Seth - Fruits and Vegetables Image Recognition Dataset (https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition)
- Shreya Maher - Fruits Dataset (Images) (https://www.kaggle.com/datasets/shreyapmaher/fruits-dataset-images)
- Israk Ahmed & Shahriyar Mahmud Mamun - Vegetable Image Dataset (https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset)
- Marko Škrjanec - Fruit Image Dataset (https://www.vicos.si/resources/fids30/)
- Saeed Anwar et al. - Natural Color Dataset (NCD) (https://www.kaggle.com/datasets/tyrionlannisterlzy/natural-color-dataset-for-colorization-task)

7.2. Dropout comparison with lightweight Turing Test

	WITHOUT DROPOUT				BIG DROPOUT			LOW DROPOUT									
	10 epochs	14 epochs	20 epochs	25 epochs	5 epochs	10 epochs	15 epochs	10 epochs	20 epochs	30 epochs	40 epochs	50 epochs	57 epochs	60 epochs	70 epochs	80 epochs	
apple	20,00%	0,00%	0,00%	0,00%	0,00%	0,00%	20,00%	0,00%	0,00%	0,00%	0,00%	0,00%	20,00%	20,00%	20,00%	0,00%	apple
banana	60,00%	80,00%	80,00%	60,00%	0,00%	20,00%	20,00%	40,00%	40,00%	80,00%	60,00%	80,00%	80,00%	80,00%	80,00%	80,00%	banana
bell pepper	20,00%	20,00%	40,00%	0,00%	0,00%	0,00%	20,00%	0,00%	20,00%	40,00%	20,00%	20,00%	40,00%	40,00%	20,00%	20,00%	bell pepper
brocoli	0,00%	80,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	20,00%	40,00%	80,00%	60,00%	100,00%	100,00%	100,00%	100,00%	brocoli
cabbage	20,00%	20,00%	0,00%	20,00%	0,00%	0,00%	0,00%	0,00%	20,00%	40,00%	40,00%	40,00%	40,00%	40,00%	40,00%	40,00%	cabbage
carrot	60,00%	40,00%	40,00%	60,00%	0,00%	0,00%	0,00%	40,00%	0,00%	60,00%	40,00%	60,00%	60,00%	40,00%	60,00%	60,00%	carrot
cherry	20,00%	60,00%	60,00%	80,00%	0,00%	20,00%	40,00%	40,00%	40,00%	60,00%	80,00%	60,00%	80,00%	80,00%	60,00%	60,00%	cherry
chilly pepper	20,00%	0,00%	40,00%	20,00%	0,00%	0,00%	20,00%	20,00%	40,00%	20,00%	20,00%	40,00%	60,00%	60,00%	20,00%	40,00%	chilly pepper
corn	20,00%	20,00%	0,00%	0,00%	0,00%	0,00%	20,00%	0,00%	0,00%	20,00%	0,00%	0,00%	20,00%	20,00%	20,00%	20,00%	corn
cucumber	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	cucumber
eggplant	40,00%	60,00%	60,00%	0,00%	60,00%	20,00%	40,00%	40,00%	40,00%	60,00%	80,00%	60,00%	80,00%	80,00%	40,00%	80,00%	eggplant
grap	40,00%	60,00%	40,00%	20,00%	0,00%	40,00%	20,00%	60,00%	40,00%	60,00%	60,00%	80,00%	80,00%	80,00%	40,00%	60,00%	grap
kiwi	20,00%	20,00%	20,00%	0,00%	20,00%	20,00%	20,00%	0,00%	0,00%	0,00%	0,00%	20,00%	60,00%	60,00%	40,00%	40,00%	kiwi
lemon	60,00%	40,00%	40,00%	40,00%	0,00%	20,00%	20,00%	20,00%	60,00%	40,00%	60,00%	80,00%	40,00%	80,00%	60,00%	40,00%	lemon
mango	60,00%	60,00%	60,00%	40,00%	0,00%	0,00%	20,00%	40,00%	40,00%	40,00%	60,00%	100,00%	100,00%	100,00%	100,00%	80,00%	mango
orange	40,00%	40,00%	40,00%	20,00%	0,00%	0,00%	0,00%	0,00%	20,00%	20,00%	20,00%	40,00%	40,00%	40,00%	60,00%	40,00%	orange
pear	20,00%	40,00%	60,00%	20,00%	0,00%	20,00%	20,00%	20,00%	40,00%	40,00%	60,00%	40,00%	60,00%	80,00%	40,00%	60,00%	pear
pinapple	60,00%	40,00%	60,00%	60,00%	0,00%	40,00%	0,00%	20,00%	40,00%	40,00%	80,00%	80,00%	80,00%	80,00%	40,00%	40,00%	pinapple
pomgrenate	20,00%	20,00%	20,00%	40,00%	0,00%	0,00%	20,00%	0,00%	0,00%	20,00%	40,00%	40,00%	60,00%	20,00%	60,00%	40,00%	pomgrenate
potato	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	potato
raddish	0,00%	20,00%	0,00%	20,00%	0,00%	20,00%	0,00%	20,00%	20,00%	20,00%	20,00%	20,00%	20,00%	20,00%	20,00%	20,00%	raddish
strawberry	20,00%	0,00%	20,00%	0,00%	20,00%	0,00%	20,00%	0,00%	20,00%	0,00%	0,00%	0,00%	40,00%	20,00%	20,00%	0,00%	strawberry
tomato	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	tomato
watermelon	0,00%	40,00%	20,00%	20,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	40,00%	20,00%	20,00%	40,00%	20,00%	20,00%	watermelon
	25,83%	31,67%	29,17%	21,67%	4,17%	9,17%	13,33%	15,00%	20,83%	29,17%	35,83%	39,17%	49,17%	49,17%	40,00%	39,17%	

7.3. ConvBlock with big Dropout

```
ConvBlock(1, 64, dropout_prob=0.1),
ConvBlock(64, 128, stride=2, dropout_prob=0.2),
ConvBlock(128, 256, stride=2, dropout_prob=0.3),
ConvBlock(256, 512, dilation=2, dropout_prob=0.4),
ConvBlock(512, 512, dilation=4, dropout_prob=0.5),
UpConvBlock(512, 256, dropout_prob=0.4),
ConvBlock(256, 256, dropout_prob=0.3),
UpConvBlock(256, 128, dropout_prob=0.3),
ConvBlock(128, 128, dropout_prob=0.2),
nn.Conv2d(128, 313, kernel_size=3, padding=1),
```

7.4. ConvBlock with low Dropout

```
ConvBlock(1, 64, dropout_prob=0.05),
ConvBlock(64, 128, stride=2, dropout_prob=0.1),
ConvBlock(128, 256, stride=2, dropout_prob=0.2),
ConvBlock(256, 512, dilation=2, dropout_prob=0.25),
ConvBlock(512, 512, dilation=4, dropout_prob=0.3)
UpConvBlock(512, 256, dropout_prob=0.2),
ConvBlock(256, 256, dropout_prob=0.15),
UpConvBlock(256, 128, dropout_prob=0.1),
ConvBlock(128, 128, dropout_prob=0.05),
nn.Conv2d(128, 313, kernel_size=3, padding=1),
```