

# **ColorUNet: A convolutional classification approach to colorization**

**CS231n Final Project, Stanford University**

Vincent Billaut  
Department of Statistics  
Stanford University  
vbillaut@stanford.edu

Matthieu de Rochemonteix  
Department of Statistics  
Stanford University  
mderoche@stanford.edu

Marc Thibault  
ICME  
Stanford University  
marcthib@stanford.edu

## **Abstract**

*This paper tackles the challenge of colorizing grayscale images. We take a deep convolutional neural network approach, and choose to take the angle of classification, working on a finite set of possible colors. Similarly to a recent paper, we implement a loss and a prediction function that favor realistic, colorful images rather than “true” ones.*

*We show that a rather lightweight architecture inspired by the U-Net, and trained on a reasonable amount of pictures of landscapes, achieves satisfactory results on this specific subset of pictures. We show that data augmentation significantly improves the performance and robustness of the model, and provide visual analysis of the prediction confidence.*

*We show an application of our model, extending the task to video colorization. We suggest a way to smooth color predictions across frames, without the need to train a recurrent network designed for sequential inputs.*

## **Introduction**

The problem of colorization is one that comes quickly to mind when thinking about interesting challenges involving pictorial data. Namely, the goal is to build a model that takes as an input the greyscale version of an image (or even an actual “black and white” picture) and outputs its colorized version, as close to the original as possible (or at least realistic, if the original is not in colors). This problem is complex and interesting for several reasons, as the final output needs to be an image of the same dimension as the input image. We want to train a model that is able to recognize shapes that are typical of a category of items and apply the appropriate colorization.

One clear upside to this challenge is that any computer vision dataset, and even any image bank really, is a proper dataset for the colorization problem (the image itself is the

model’s expected output, and its greyscale version is the input to the model). The input to our algorithm is simply an image in greyscale, and the output is the same image, colorized. A conversion of the images to the YUV format allows an easy formulation of the problem in terms of a reconstitution of the U and V channels<sup>1</sup>.

We formulate the colorization problem as a classification problem to gain flexibility in the prediction process and output more colorful images. We aim at reproducing state of the art results that give vivid, visually appealing results, with a much smaller network.

One of our main requirements is designing our own architecture from scratch, rather than using parts from other architecture or pre-trained portions, to ensure modularity, full control of the design and keep the overall complexity reasonable.

## **1. Related Work**

Historically, older approaches of the colorization problem use an additional input, a *seed scribble* from the user to propagate colors, as does [8]. It may also be seen as a corollary of color style transfer algorithms using a similar image as a “seed” as in [5]. Both approaches yield very good results but need supervision from the user for each image. In this project, we are interested in fully automatic recolorization.

Classical approaches to this task, *e.g.* [2] and [3], aim at predicting an image as close as possible to the ground truth, and notably make use of a simple  $L_2$  loss in the YUV space, which penalizes predictions that fall overall too far from the ground truth. As a consequence, the models trained following such methods usually tend to be very conservative, and to give desaturated, pale results. Usually, the images need some postprocessing adjustments as in [4] to have a realistic

---

<sup>1</sup>The transformation between RGB and YUV color encodings is a linear transformation. In the YUV system, Y is the luminance (greyscale) and U,V encode the colors.

aspect.

On the contrary, authors of [12] take another angle and set their objective to be “*plausible* colorization” (and not necessarily *accurate*), which they validate with a large-scale human trial. To achieve such results, they formulate the colorization task as a classification task using color bins, as suggested already in [1].

Their approach is the most appealing as they have colorful results and we found the classification formulation to be interesting. However, the network architecture they use is heavy (constraining to small batch sizes to fit into a single GPU), and the scale of their training set (several millions of images) is prohibitive. The reason behind this complexity is that in order to encode meaningful features that help to colorize the image, the receptive field has to be large. The approach of the article is to downsample the image a lot in the intermediate layers and then upsample it using Transpose Convolution layers. To keep a lot of information in the intermediate layers, the number of filters in the model of [12] has to be significant, resulting in a large, slow and expensive training.

Authors of [9] have shown that connections between hidden layers of a bottleneck neural network could enhance the performance greatly, by injecting locational information in the upsampling process, and improving the gradient flow.

We hope that applying this method will allow us to train a colorizing model more quickly and more efficiently, with less parameters, and on a smaller dataset.

Part of the challenges that are interesting but not yet tackled in the literature involve videos. General information propagation frameworks in a video involving bilateral networks as discussed in [6] could be seen as a good starting point to implement consistent colorization of video sequences, if we manage to embed the colorizing information and then propagate it as any other information. The work realized in [13] is also interesting since it tackles video stylization by grouping the frames, choosing a representative frame for each group and using the output of the network on this frame as a guideline, which enhances temporal consistency. However, the adaptation of such an algorithm to the much more complex task of image colorization is far beyond the scope of this project.

Actually, one promising way to perform image colorization is to be able to learn meaningful color-related representations for the images (which often involves using very deep and heavy or pretrained architecture as in [7]) and then ensure the temporal consistency of them.

Given our commitment to developing a lightweight model, we prefer focusing on an efficient colorization for images and then add a layer of temporal convolution to stabilize the video colorization.

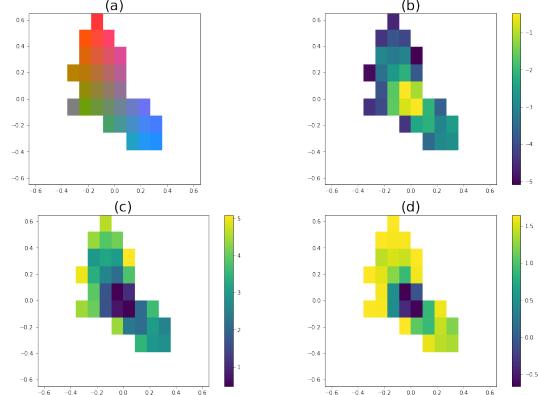


Figure 1. (a) Color map, showing the mean color (*chrominance*) of each of the selected bins (here, we set a threshold of 32 bins to select). (b) Frequency map (log-scale), shows the empirical frequency of the colors within each bin, computed over a dataset beforehand. (c) Inverse-frequency map (log-scale), *i.e.* the inverse of (b). (d) Weight map (log-scale), shows the weights assigned to each bin after rebalancing. Interestingly, we notice that the amplitude in weights is much smaller than the amplitude in frequency (2 orders of magnitude against 4), which means that we partially make up for the underrepresentation bias and therefore encourage the prediction of rare colors.

## 2. Methods

### 2.1. Colorization as classification

As we discussed in section 1, we are taking the angle of [12]. As a result, we are approaching the colorization as a classification problem, and therefore using a (weighted) cross-entropy loss.

Concretely, we want to discretize our colorspace, and for that we simply split our colormap into equal sized bins. As a first step, in order to reduce the computational toll, we don’t want too many bins, and are therefore restricting our discretization to the  $n$  most frequent color bins, as learned on a big dataset beforehand. In what follows, if a color from our actual image does not fall within one of our  $n$  bins, we will simply assign it to the closest available. This simplification leads to a rather faint degradation of the images.

Following the approach of [12], we want to boost the possibility of a rare color being predicted, and therefore reproduce the following tricks:

- Use *rebalancing* to give larger weights to rare colors in our loss function. Precisely, each pixel value  $p$ , assigned to its closest bin  $b \in \mathbb{R}^n$  is given a weight  $w_p$  such that

$$w_p \propto \left( (1 - \lambda) \hat{P}(b) + \frac{\lambda}{n} \right)^{-1}$$

where  $\hat{P}(b)$  is the estimated probability of the bin

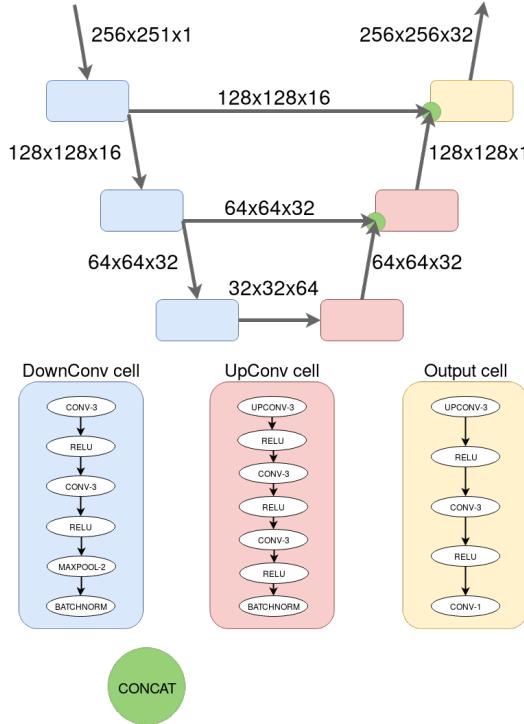


Figure 2. Structure of the *ColorUNet*. We use 3 types of cells: DownConv Cells that use 2 stacked convolutional layers to have a large receptive field and a maxpooling to downsample the image, UpConv cells that use 1 ConvTranspose Layer to upsample the image and then 2 convolutional layers, and an Output cell that is a simplified version of the UpConv cell. All the convolutional layers have kernel size  $3 \times 3$  with stride 1 and padding 1. The ConvTranspose layers have kernel size  $3 \times 3$  with stride 2 and padding 1. The maxpooling layers have size  $2 \times 2$  and stride 2, with no padding.

(computed prior to our model’s training) and  $\lambda \in [0, 1]$  a tuning parameter (the closer to 1, the less we take the bin’s frequency into account).

- Use an *annealed-mean* to output predictions  $y$  from the probability distribution  $\mathbf{Z}$  over our  $n$  bins to the full original color space. The idea is to find a compromise between taking the color class with the maximum probability (the mode), which gives a rather colorful result but sometimes lacking spatial consistency, and taking the weighted average over the bins, which gives a rather flat, sepia kind of tone. To achieve this we use a temperature parameter  $T \in (0, 1]$  in the following softmax-like formula for one pixel

$$y = f_T(\mathbf{z}) = \frac{\exp(\log(\mathbf{z})/T)}{\sum_i \exp(\log(\mathbf{z}_i)/T)}$$

where  $\mathbf{z}$  is the  $n$ -dimensional probability vector of a given pixel over the  $n$  bins, and the sum in the denominator is over all the bins.

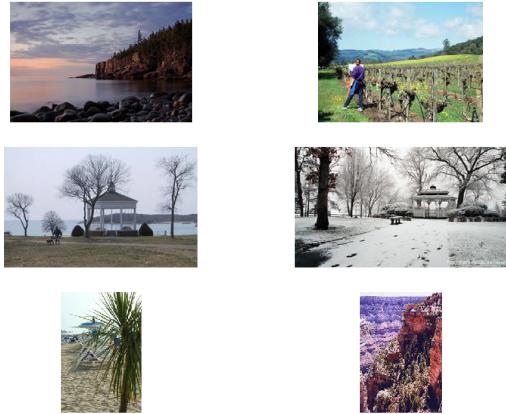


Figure 3. Sample images from our dataset (here, the validation set). We included categories such as coast, beach and gazebo\_exterior from SUN, and categories like landscape, field and canyon from ImageNet.

Figure 1 shows our discretized colorspace, as well as what the weights of the bins look like after rebalancing.

Our task has now become a segmentation problem, for which we have to predict the correct class (representing a color bin) out of 32, for every pixel in the input image. Our end model is a U-Net that has 3 downsampling groups of convolutional layers and 2 connections between hidden layers of the same size (we set aside the last and first layers of the downsampling process).

Figure 2 summarizes the structure of the final network, that we refer as *ColorUNet*.

To find this final model, we have tried several architectures. We have begun by trying a simple “flat” convolutional network with 3 layers. However, this model gave very poor performance. Intuitively, what happens is that the spatial receptive field was not large enough to capture meaningful general shapes. Starting from this idea, we then implemented a second model, that is deeper, with 6 groups of convolutional layers. The structure is a *bottleneck*, where 3 groups of layers downsample the image using a max pooling, and 3 groups of layers upsample the image. Using groups of  $3 \times 3$  convolutional layers rather than bigger filters allows to have a large receptive field with fewer parameters. The *bottleneck* structure is a way to constrain the model to embed recognizable object shapes and the associated color.

This network gave interesting results, but it was very unstable and slow to train. This is easily explained by the depth of the architecture. Furthermore, the quality of the output we get was not satisfying, even for a reduced task, because of this underfitting. Furthermore, it would have needed more filters in the hidden layers to tackle the up-sampling part of the flow, as we need to encode more information.



Figure 4. Sample predictions of the ColorUNet on the validation set, for several prediction temperatures. The temperature parameter allows a tradeoff between vivid colors and elimination of artifacts of a wrong color.

mation in the smaller layers to correctly upsample the image afterwards.

Those observations led us to design our final *ColorUNet*, that still has a bottleneck structure but uses connections between layers to improve the gradient flow and help the up-sampling. We also added additional batch normalization layers to improve training stability.

## 2.2. Neural architecture: *ColorUNet*

## 2.3. Datasets

## 3. Dataset and Features

To train our model, we used subsets of the SUN [11] and ImageNet [10] datasets. We selected 8 categories from ImageNet and 14 categories from SUN, that correspond mainly to nature landscapes. Our final training set is composed of 13,469 images – as a reference for comparison, [12] train their model on 1.5M+ images. Our validation set is made of 2,889 images. A sample of the training data is shown in Figure 3. The motivation behing reducing our focus to nature scenes was both conceptual and practical. Conceptual because it seems too optimistic to try to train a rather lightweight model on a very diversified dataset, and practical because it eventually comes down to computing time. We chose to include images representing nature scenes which have pictoral elements in common – such as trees, mountains, clouds, *etc.* – as well as color tones – blue for the sky, sand for beaches, green for grass and forests,

*etc.*

We also tried training the model on the full SUN dataset – roughly 100k images –(for only one epoch), to compare performance and gain insight on the importance for the model to go over the same examples several times to learn well.

To keep a reasonable size for our tensors and have uniformity in our dataset, the images have all been downsampled to fit in a  $256 \times 256$ px frame (if downsampling was necessary). The downsampling has been performed using Lanczos method (with a *sinc* kernel), and we used a mask for non-square images to make sure the loss is relevant.

## 3.1. Data Augmentation

To improve the robustness of the model, we augmented our training data with a number of deformations. For each original image of the training set, we generate several training images by combining:

- Flipping the image along the horizontal axis;
- Adding noise to the image (with different intensities);
- Selecting random crops of the image.

We expect an improvement of training stability and of the quality of our predictions when including the data augmentation in the pipeline, compared to when simply using more images for training. In the case described above, the size of our training set is dilated sevenfold. We discuss the effect of data augmentation in section 4.



Figure 5. Sample predictions of the ColorUNet on the validation set, for bland input images. The ColorUNet’s output is more colorful than the ground truth. The bottom example is an old photograph with weared tones.

## 4. Results and discussion

### 4.1. Evaluation and parameter tuning

Given the task that we are performing here, there is no relevant metric of the performance of our model. Actually, the classification loss is useful for training but is not very readable in terms of evaluating the actual prediction performance. Likewise, another loss like the  $L_2$  is not relevant, as we discussed in section 1. Similarly to [12], we rely on human evaluation to evaluate the overall performance of the algorithm, as it is both quick and easy to have a look at a random subsample of the validation set and see if the coloring is satisfying.

The final number of classes chosen for the ColorDiscretizer is  $n = 32$ , as we want this to be as small as possible (it conditions the size of the output tensor) but 30 color bins is enough to get a very good visual fidelity.

We have chosen the size of our network layers (especially the number of filters) to be of a reasonable order of magnitude, *i.e.* around 32, as our focus was to have a computationally reasonable model. To select the learning rate, we have observed the evolution of the loss on reduced epochs and kept a learning rate that showed a satisfying behavior for the loss. We used an Adam optimizer. Our best model has been trained in 2 steps, with a learning rate decay between the 2 steps. Finally, we have chosen a batch size of 64 as it was the largest to fit on the GPUs we had, as the output tensor has dimensions  $256 \times 256 \times 32 \times \text{batch\_size}$  (We used a single NVIDIA Tesla K80 GPU).

Finally, we have used a validation set to evaluate the model. The split between validation and training has been done randomly at preprocessing time.

In addition to visualizing our own colorization results,

we have visualized the results of the state-of-the-art model, [12], that is available online. Not surprisingly, the results are more realistic. However, on a lot of cases from our validation set, the results are similar. More interestingly, we see that errors are **visually similar** for the two networks, *i.e.* color stains and that the same kinds of objects cause errors (textures of grass that look like waves on the ocean *e.g.*). This provided a general sanity check for the behavior of our model.

### 4.2. Colorizing images

#### 4.2.1 Overview of the results

Figure 4 gives some examples of correct colorizations made by the ColorUNet. We can see that the network is able to correctly colorize the main entities of interest (The sky, trees, grass, water, clouds, etc...) with vivid colors. The influence of the temperature parameter is very easy to see here, as we have less consistency but more vivid colors for low temperatures (closer to taking the argmax) and smoother, but desaturated for higher temperatures, as expected in section 2.

Interestingly enough, the ColorUNet sometimes gives results that are not similar to the input image, but look better. Those examples are interesting, because they reveal how the coloring model works. Some of them are reported in Figure 5. This happens with images that have very recognizable inputs, but an limited color palette. The ColorUNet output is more realistic in those cases. Such examples also show that it is difficult to quantitatively evaluate the model. Indeed, any reasonable computational metric would suggest that these outputs are wrong, while this is clearly the kind of results we are especially interested in.

On the other hand, there are examples where our network is obviously wrong and outputs predictions that are

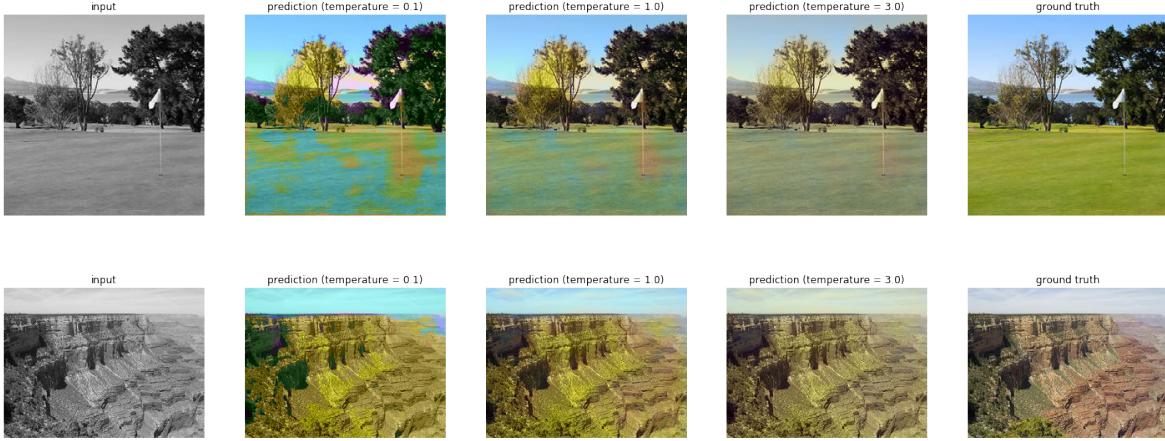


Figure 6. Sample bad predictions of the ColorUNet. On the top picture, the network is fooled by the texture of the grass, that has small wavelets similar to sea landscapes, that constitute a vast portion of the training set. On the bottom picture, the network predicts the canyon as vegetation. It has not learned the color to canyons, because there are too few in the training set, and because the color palette for those landscapes is very specific.

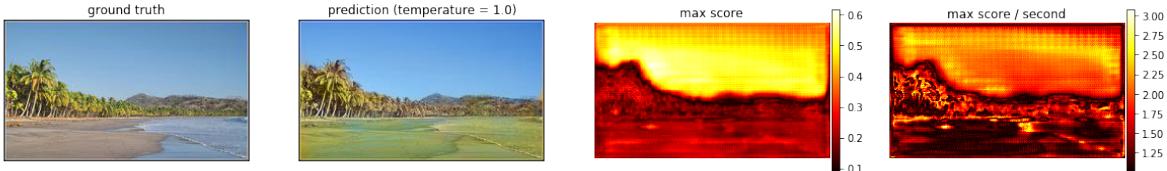


Figure 7. Confidence map for bad predictions of the ColorUNet. We plot the original image, the prediction with a chosen level of temperature, the value of the top class score, and the ratio between the first class score and the second one to each pixel to get a confidence map. Dark/red pixels correspond to unsure predictions, whereas yellow/light are associated with a high confidence score. The network is fooled by the texture of the sand and sea, that it predicts as grass. On the other hand, the sky is correctly predicted. The confidence maps show that the ColorUNet is sure of its prediction on the sky, but is unsure of the prediction for the ground. It is also interesting to note that the regions close to the edges are often more uncertain, although the network performs very well in segmenting parts of the image that should have different colors.

not plausible. Some such examples are seen in 6.

Overall, our ColorUNet performs well at the limited task that we want to tackle. Not very surprisingly, it does not perform well on new categories, and especially on inside scenes.

#### 4.2.2 Further analysis

We propose some visualizations to further analyze the behavior of the model.

**Confidence maps** In order to better understand the behavior of our model, we have designed a **confidence** map, showed in Figure 7. We use the value of the top bin’s score and its relative magnitude compared to the second score of each pixel to visualize the confidence of the prediction. This is also a good sanity check. Actually, given that our model is intended to be lightweight, we do not expect it to make

outstanding predictions on all the pictures. However, we expect our model to be unsure about the wrong results, and the difficult parts of the images.

**Loss analysis** The ability of the model to generalize can also be explored from a quantitative point of view. Figure 8 shows that the performance of our model on the validation set matches that of the training set, in terms of loss. We have avoided overfit, so that this result validates our initial idea of aiming at a lightweight model, which can not be over-parametric. This is also promising since it implies that there is room for progress. The *ColorUNet* can be further trained, and kernel sizes in intermediate layers can be increased, which will yield even better results with more computing power.

When having a closer look at in-sample examples, we see that overall, the quality of the prediction is slightly better but comparable to what we have in the validation set.

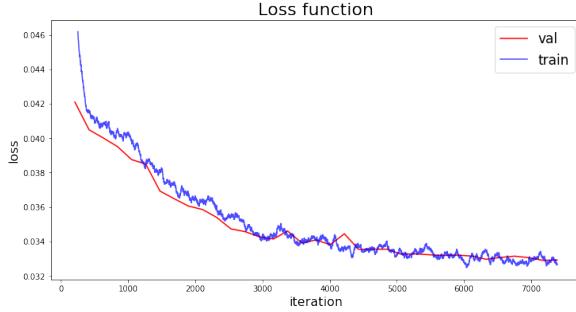


Figure 8. Evolution of the loss of *ColorUNet* accross the learning iterations. The validation (red) loss matched the train (blue) loss, so that we have not overfitted our training set.

This suggests that we have not overfitted the training set. However, given the evolution of the loss, we can even expect our model to be slightly under-fitted, and it may benefit from further training. However, given the limited time we had to complete this project, we preferred focusing on other experiments (such as studying the effect of data augmentation or training on an extended dataset) rather than spending time overfitting a model.

**Color distribution analysis** In order to understand the impact of our decision to encourage the prediction of "rare" colors by weighting the loss, we produced in Figure 9 histograms of the frequencies of the 32 color bins. We see that the prediction of the *ColorUNet* does not exactly match the ground truth. By comparing the histograms with the frequencies in the whole dataset, we see that the ground truth had "common" colors. The *ColorUNet* chooses to aim for "rare" colors, thus producing vivid images. Since this choice does not impair the realism of outputs, we believe that our contribution achieved our goal of colorizing with lively colors.

**Influence of data augmentation** To evaluate the influence of the data augmentation, we have run 2 models on the same training set, with and without data augmentation, with seven times more epochs without data augmentation, so that both training have the same number of iteration. As expected, this brings more stability to the predictions, especially for the low temperatures.

Figure 10 compares the results of those two experiments.

### 4.3. Application to video colorization

The last task we tried to tackle is colorizing greyscale videos. The natural way to tackle this problem is to extract individual frames that constitute the video, and to every frame with the previous *ColorUNet*. Converting this series of colorized images back into a video produces a col-

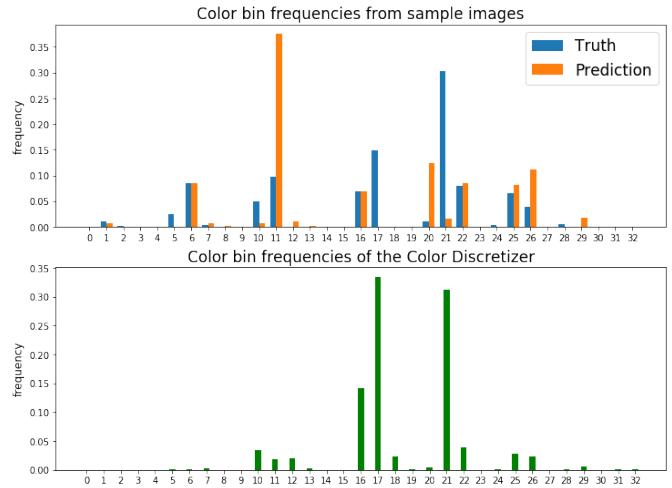


Figure 9. Histograms of the frequencies of the 32 selected color bins. Top: frequencies in the pixels of 500 sample images; true color channels (blue); colors predicted by *ColorUNet* (orange). Bottom: frequencies of the color bins in the Color Discretizer (green), representing the whole dataset. The *ColorUNet* differs from the ground truth which has "common" colors (bins 17, 21), it prefers "rare" colors (bins 11, 20, 26).

orized video. The results of this work can be seen on our Github repo's homepage.

One understands here that this preliminary work does not incorporate the structure of the video. However, this information might be crucial for two reasons. First, as the movie evolves over a progressively moving scene, we want some stability over the set of colors that are predicted. From one frame to the next, the model should colorize in a similar way the corresponding items. Second, an algorithm having an extra temporal information on top of the single-frame prediction might even perform better. Indeed, including past information enables understanding better the context of the image, and incorporate content that is now hidden from the frame.

Consequently, the method we will use should know what was predicted before and take this knowledge into account when deciding for the next frame's color. The choice we made, both to stabilize predictions and to incorporate past information, is smoothing the predictions by applying a temporal kernel, which weights the past predictions' probabilities with an exponential decay:

$$\hat{p}_t = \sum_{i=0}^T p_{t-i} e^{-\alpha i}.$$

The 32-class probability vector we use for the video colorization,  $\hat{p}_t$ , aggregates past frames' probabilities  $p_{t-i}$ , over  $T = 20$  frames, with a decay coefficient of  $\alpha = 0.2$ . The results are smoother and somehow more natural.



Figure 10. Influence of data augmentation. Top: prediction for 2 levels of temperature, with data augmentation. Bottom: prediction for 2 levels of temperature, without data augmentation. The data augmentation tends to predict finer and smaller color sains that are closer to the actual coloring, which is easy to see on the low temperature prediction on the left. This results in an overall enhancement of the final prediction (on the left), where the sky tones are more realistic

#### 4.4. Extending the training set

As a sanity check, we have also trained a model using the full SUN dataset, that is 10 times bigger than our actual training set, with data augmentation. For time constraints reason, we have trained it on fewer epochs. The training loss was similar. However, we note that the result were very poor, and that the network tended to output very desaturated images. Actually, our network structure has few filters in its deepest layers compared to the baseline model, [12]. It is therefore unable to capture as many features as needed to be able to recognize the variety of scenes that are in the full dataset. This confirms that our architecture is a lightweight version, that is adapted to a reduced variety of scenes, and that trains rapidly. However, we could expect that applying the same structure with more filters, to have a complexity closer to [12] could allow to have a much quicker training on a full dataset.

An interesting next step to take would be to do a full benchmark of the model performance over different ranges of variety in the training set. Actually, our lightweight model does not perform well on many classes because it is not complex enough. However, too few classes does not allow to state an interesting problem. For example, learning on beaches can be done with bimodal predictions (blue or yellow). An extensive quantitative benchmark of this effect could be interesting but very time and resource-consuming, and beyond the scope of this project.

## 5. Conclusion and perspectives

In this work, we formulated the colorization problem as a classification problem to get colorful results. The ColorUNet we have designed is a lightweight version of the state-of-the-art architecture that allows to quickly get satisfying results on a reduced dataset, with limited computational power.

Our predictions have similar limitations as the reference model. Extending the task to more categories would need more parameters, but our experiments has shown that a U-Net inspired architecture enhances and accelerates the learning process for the colorization problem. We propose a lightweight extension to colorize video sequences.

Possible extensions would include training a network with more filters on more classes, but it would most likely yield similar results to [12], and still have consistency problems. Using the confidence maps we designed to automatically select seed pixels and apply the heuristics described in [8] to *diffuse* those seeds would be an elegant way to post-process our images and have realistic, consistent but yet colorful results, taking the best of "traditional" and "convolutional" approaches to colorization.

Using *seed pixels* from our predictions and diffusing them could also be a way to adapt the heuristics of [13] to colorization, by enforcing temporal consistency of the seed pixels as a proxy to enforce temporal consistency for the colorization.

## Contributions and acknowledgements

We would like to thank the CS231n teaching staff for giving us access to the Google Cloud Compute platform and for the teaching material.

All authors contributed equally to this work.

## References

- [1] G. Charpiat, M. Hofmann, and B. Schölkopf. Automatic image colorization via multimodal predictions. In *European conference on computer vision*, pages 126–139. Springer, 2008.
- [2] Z. Cheng, Q. Yang, and B. Sheng. Deep colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 415–423, 2015.
- [3] R. Dahl. Automatic colorization. <http://tinyclouds.org/colorize/>, 2016.
- [4] A. Deshpande, J. Rock, and D. Forsyth. Learning large-scale automatic image colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 567–575, 2015.
- [5] M. He, J. Liao, L. Yuan, and P. V. Sander. Neural color transfer between images. *CoRR*, abs/1710.00756, 2017.
- [6] V. Jampani, R. Gadde, and P. V. Gehler. Video propagation networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2017.

- [7] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. *CoRR*, abs/1603.06668, 2016.
- [8] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. In *ACM Transactions on Graphics (ToG)*, volume 23, pages 689–694. ACM, 2004.
- [9] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.
- [10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [11] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.
- [12] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.
- [13] F. Zhu, Z. Yan, J. Bu, and Y. Yu. Exemplar-based image and video stylization using fully convolutional semantic features. 26:3542–3555, 07 2017.