Image Colorization Using Convolutional Neural Networks

SHRISH R
2020506088

DEEPAK ATHIPAN AMB
2020506021

JEFFREY SAMUEL S
2020506038

Abstract. Colorization is a process of converting grayscale images into visually acceptable color images. Automatic conversion has become a challenging area that combines machine learning and deep learning with art. Interfaces which the user can easily access and interact with to color the greyscale images are even rarer. Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. Convolutional Neural Networks in Deep Learning can be used to analyze and color the grayscale images. CNNs are particularly useful for finding patterns in images to recognize objects, classes, and categories. A CNN model built from scratch as well as an autoencoder model and a pretrained model were all trained on the Flickr 8k Image dataset and validated using a 20% validation set. The best model has been chosen and a Python-Flask backend along with a HTML-CSS designed website has been constructed for improved human computer interaction and better access for image colorization.

1 Introduction

Image colorization is the process of adding color to grayscale or black-and-white images. It involves utilizing image processing algorithms and machine learning techniques to infer the most appropriate color for each pixel in the image. The goal of image colorization is to produce a color version of a grayscale or black-and-white image that is visually plausible and consistent with the original image. This process can be used to enhance old photographs or film footage by adding color, which can breathe new life into historical images and make them more accessible to modern viewers. It can also be used in modern applications such as computer vision and image recognition.

Image colorization is a challenging problem, as there is often a wide range of possible colors that could be applied to each pixel, and the algorithm must determine the most appropriate color based on the context of the image. Various techniques have been developed to address this problem, including deep learning-based methods, which use neural networks to learn the mapping between grayscale images and their corresponding color images.

In recent years, image colorization has become an active area of research in computer vision and has led to the development of several state-of-the-art algorithms. These algorithms have the potential to revolutionize the way we process and analyze images, and they have already been used in a wide range of applications, including photo and video editing software, image and video search engines, and digital restoration of old images and films.

Image colorization is a challenging task in computer vision that aims to generate color images from grayscale images. Image colorization has many applications, such as in the restoration of old photographs and the generation of realistic images. Traditional methods for image colorization rely on manual colorization or semi-automatic methods that require user input. However, these methods are time-consuming and require a high degree of expertise. In recent years, deep learning methods have shown promising results in image colorization tasks. In this paper, we present a deep convolutional neural network for automatic image colorization which has been trained using the Flickr 8K dataset.

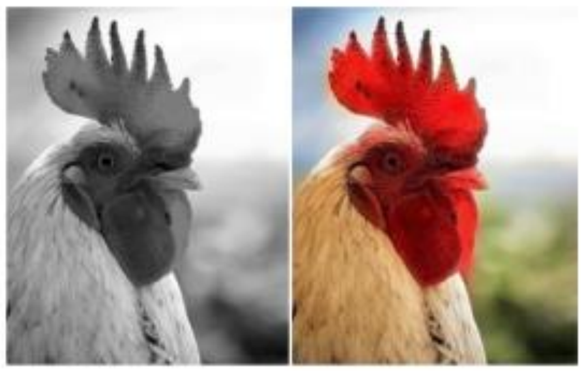


Figure 1 An example of image colorization

The main idea behind CNN-based colorization is to train a neural network to learn the mapping between grayscale images and their corresponding color images. We compare the CNN model with a pre-trained CNN model and an auto encoder model as well. The best model is analyzed and chosen and a web-app has been implemented using Flask and HTML, CSS and Python.

2 AUTOENCODER MODEL

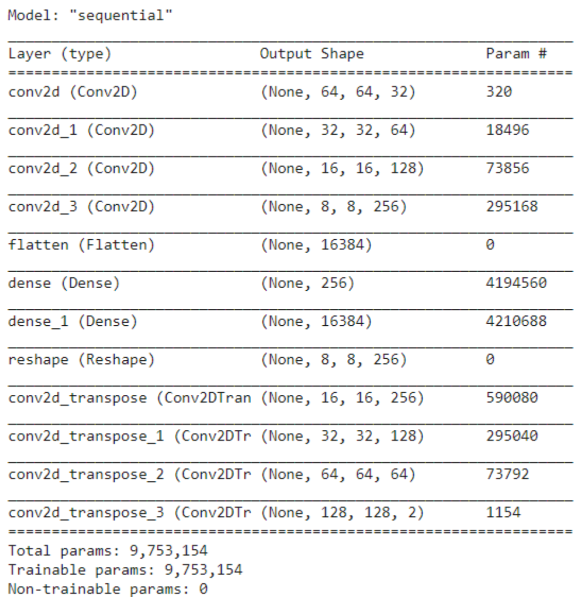
An autoencoder is a type of neural network that learns to compress and reconstruct input data. In image colorization, the autoencoder is trained to map a grayscale input image to its corresponding color image.

The architecture of the model consists of two parts: an encoder and a decoder. The encoder consists of four convolutional layers that reduce the spatial dimensions of the input image and extract high-level features. The central layers of the model flatten the output of the encoder and pass it through two fully connected layers to compress the encoded representation of the input image. The decoder consists of four transposed convolutional layers that increase the spatial dimensions of the compressed encoded representation and generate the final color image. The output of the decoder is a color image with two channels (one for the red-green color axis and one for the blue-yellow color axis).

The model is built using the Keras Sequential API and includes layers such as Conv2D, Flatten, Dense, Reshape, and Conv2DTranspose.

The Conv2D and Conv2DTranspose layers perform convolutional operations on the input and output feature maps, respectively. The Flatten and Dense layers are used for the central layers of the model that compress and decompress the encoded representation of the input image. The training was done using RSM prop optimizer and Mean Square Error loss.

Overall, this autoencoder model using CNNs is a powerful approach for image colorization, and the provided code can be used as a starting point for further customization and experimentation.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 256)               4194560
_________________________________________________________________
dense_1 (Dense)              (None, 16384)             4210688
_________________________________________________________________
reshape (Reshape)            (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 16, 16, 256)       590080
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 128)       295040
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 64, 64, 64)        73792
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 128, 128, 2)       1154
=================================================================
Total params: 9,753,154
Trainable params: 9,753,154
Non-trainable params: 0
```

Figure 2: Architecture of the Autoencoder model

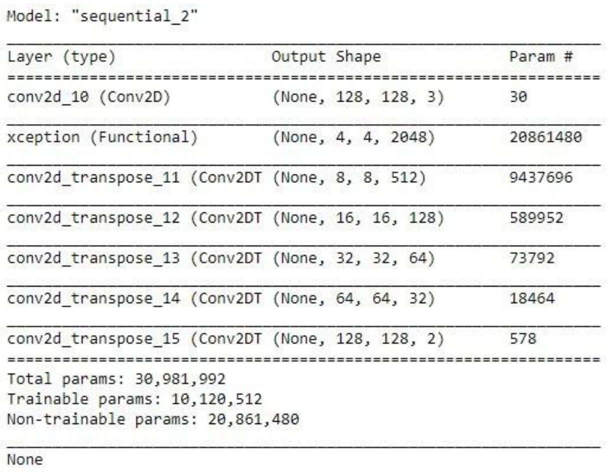
3 CNN USING PRETRAINED MODEL

The pretrained CNN model is a model for image colorization which uses a pre-trained Xception model as the encoder. Xception is a deep neural network architecture that is based on the Inception architecture and uses depth wise separable convolutions to reduce computational complexity.

The architecture of the model consists of three parts: an input layer, a pre-trained Xception model, and a decoder. The input layer transforms the grayscale input image to an RGB image. The pre-trained Xception model is loaded from the Keras Applications API and takes the RGB image as input. The decoder consists of four transposed convolutional layers that increase the spatial dimensions of the compressed encoded representation and generate the final color image. The output of the decoder is a color image with two channels (one for the red-green color axis and one for the blue-yellow color axis).

The model is built using the Keras Sequential API and includes layers such as Conv2D, Conv2DTranspose, and Xception. The Conv2D and Conv2DTranspose layers perform convolutional operations on the input and output feature maps, respectively. The Xception layer is a pre-trained model that extracts high-level features from the input image. The Conv2DTranspose layers increase the spatial dimensions of the encoded representation and generate the final color image.

The weights of the pre-trained Xception model are frozen by setting the trainable attribute of the second layer of the model to False. This ensures that the pre-trained weights are not updated during training. The model along with autoencoder model and the proposed CNN model have been compared and analyzed and the best model has been selected among them.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)          (None, 128, 128, 3)       30

_________________________________________________________________
xception (Functional)       (None, 4, 4, 2048)        20861480

_________________________________________________________________
conv2d_transpose_11 (Conv2DT (None, 8, 8, 512)        9437696

_________________________________________________________________
conv2d_transpose_12 (Conv2DT (None, 16, 16, 128)      589952

_________________________________________________________________
conv2d_transpose_13 (Conv2DT (None, 32, 32, 64)       73792

_________________________________________________________________
conv2d_transpose_14 (Conv2DT (None, 64, 64, 32)       18464

_________________________________________________________________
conv2d_transpose_15 (Conv2DT (None, 128, 128, 2)      578
=================================================================
Total params: 30,981,992
Trainable params: 10,120,512
Non-trainable params: 20,861,480
_________________________________________________________________
None
```

Figure 3: Architecture of the pretrained CNN model

4 PROPOSED CNN MODEL

The first model trained was inspired by the architecture used in [8]. The model is build of 8 blocks of convolutional layers, without any pooling layers and using only batch normalization at the end of each block, as the only regularization. The implemented model did not follow exactly the proposed architecture: it does not have a softmax activation layer towards the end of the network and it uses standard Adam optimizer and Mean Square Error loss.

It is a convolutional neural network (CNN) model built using the Keras API in TensorFlow. The model is designed for image segmentation, where the goal is to classify each pixel in an image into one of several predefined classes. The model consists of several layers of convolutional and batch normalization operations, followed by a layer for up sampling the output. The layers are defined in a Sequential model, which means that each layer is connected to the previous one in a linear fashion.

The first layer in the model is a Conv2D layer with 64 filters, a kernel size of 3x3, and a stride of (1,1). This layer also specifies an input shape and the activation parameter is set to 'relu', which means that the rectified linear unit function will be used as the activation function.

The next layers are similar Conv2D layers with different filter sizes, strides, and activation functions. Batch normalization is added after each convolutional layer to normalize the output of the layer before passing it to the next layer.

The last layers of the model are a Conv2DTranspose layer, which performs up sampling by transposing the convolutional operation, and a final Conv2D layer with 2 filters and a kernel size of 1. The UpSampling2D layer then performs additional up sampling by increasing the resolution of the output.

The proposed model uses dilated convolutions to increase the receptive field of the network, whereas the autoencoder model does not do so. Moreover, the proposed model uses transposed convolutions to up sample the output of the convolutional layers, whereas the autoencoder model uses a simple interpolation method to do so. The model is a fully conventional neural network and outputs a 2-channel image representing the chrominance values of each pixel, whereas the Xception model is a hybrid model that combines convolutional and depth wise separable convolutional layers and it outputs a 313-channel image representing the full color values. The architecture of the proposed model is shown in figure 4.

5 DATA PROCESSING

In order to train the models, the Flickr 8K dataset had to be preprocessed before being provided to the models. The original color space of the images was the infamous RGB representation, which represents the color from an image using three different channels: Red, Green and Blue. However, this color space does not approximate to the human vision, and previous literature has shown that better results can be achieved by using the LAB color space.

LAB format splits the image in 3 channels:

- L: Lightness of the image on a scale from 0 to 100. This channel represents a grayscale image.
- a: green-red color spectrum. The values range from -128 (green) to 127 (red).
- b: blue-yellow spectrum. The values range from -128 (blue) to 127 (yellow).

In the preprocessing, the L channel needs to be separated from the rest, as it will be used as the input for the model, while the combination of a*b channels will be used as ground truth for the prediction. Once all the images have been normalized, converted to LAB format and their channels separated, the dataset is divided into training and validation sets using 80:20 split. All these tasks were implemented using tf.data.Dataset API, which supports writing efficient pipelines for the input data.

6 IMPLEMENTATION

The code was implemented using Jupyter Notebook, so it can be executed using any environment that allows this file format. It can be divided in the following main sections:

- Data Preprocessing: This section covers the preprocessing applied to the input data, including the conversion from rgb image to lab, the creation of the training and validation sets, and the configuration of tensorflow data pipelines for a higher performance during training.

- Models: It contains the implementation using Keras of the three models presented in the previous section, alongside the trainable parameters for each one.

- Train Stage: One of the previous models can be selected and trained with the images found in the training set. In each epoch, the validation accuracy and loss is printed. At the end of the training, the accuracy-loss plot is displayed for both train and validation set.

- Test Stage: It retrieves one batch of the validation set and prints 4 predictions compared to their original images.

- Convert custom pictures: An image path can be added containing a black and white image and the best model is used to predict its respective image in color.

- Creation of WebApp: The best model is saved along with trained weights. Using the Flask Framework a WebApp has been implemented where the user can interact and upload greyscale images to be colored.

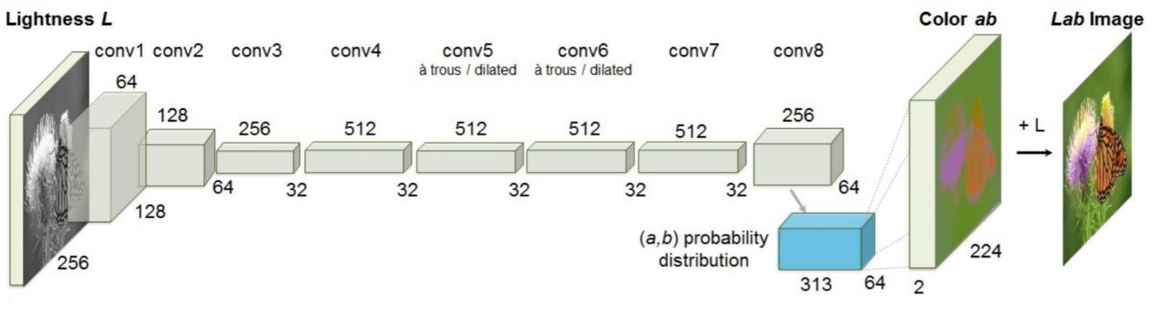


Figure 4: Architecture of the proposed CNN model

The saved model is then used to color the greyscale image and display the colored image to the user.

7 RESULTS

The results obtained for each of the above mentioned models are as follows:

CNN Model:

The model consisting of a Convolutional neural network obtained the following results in regards of the accuracy and loss both in training and validation stages after 120 epochs.

- Train Loss: 53.5244 MSE
- Train Accuracy: 80.80%
- Validation Loss: 209.4783 MSE
- Validation accuracy: 64.27%

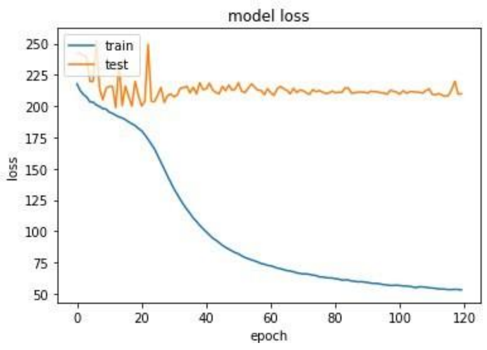


Figure 5: CNN model – Loss Chart (120 epochs)

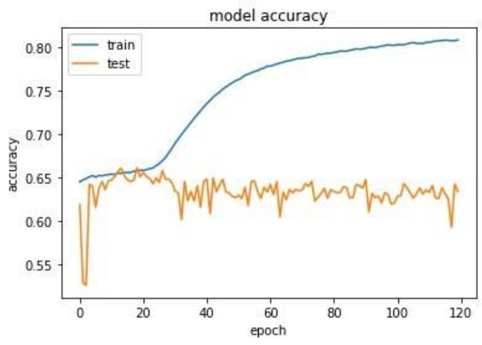


Figure 6:CNN model – Accuracy Chart (120 epochs)

The next illustration represents the results of the colorization of a random batch from the validationset:

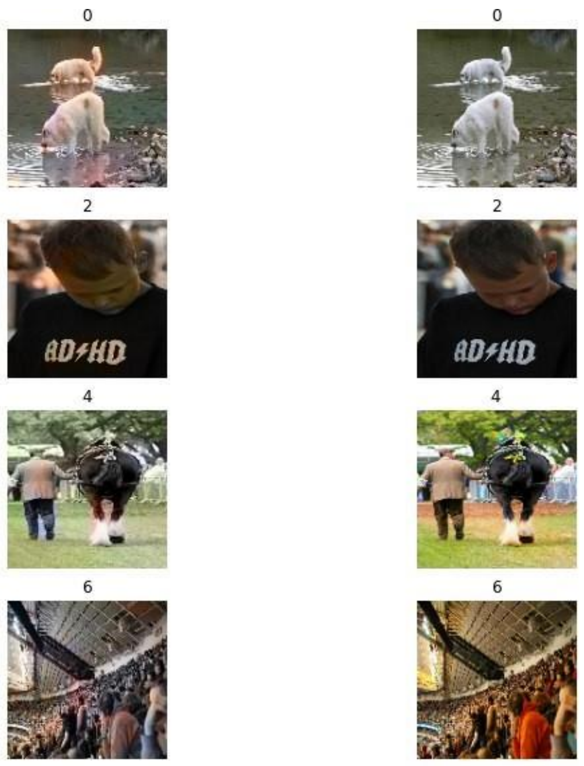


Figure 7: Original vs Colorized Images

Auto Encoder Model:

The results for the suggested auto encoder model are as follows:

- Train Loss: 92.0349 MSE
- Train Accuracy: 78.80%
- Validation Loss: 241.1622 MSE
- Validation accuracy: 63.50%

It can be seen that the autoencoder model has performed less accurately than the proposed CNN model. The next illustration represents the results of the colorization of a random batch from the validationset:

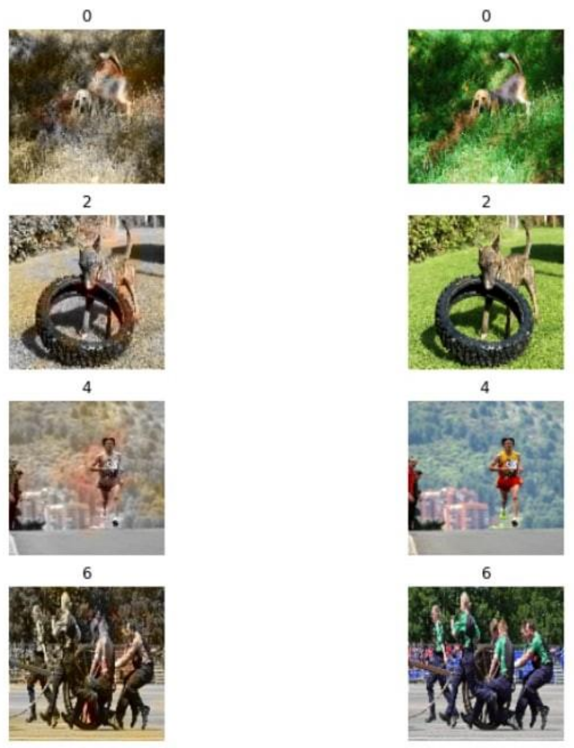


Figure 8: Original vs Colorized Images

Pre-Trained CNN Model:

The suggested pre-trained model consisting of a Convolutional neural network obtained the following results.

- Train Loss: 130 MSE
- Train Accuracy: 70.0%
- Validation Loss: 155 MSE
- Validation accuracy: 68.7%

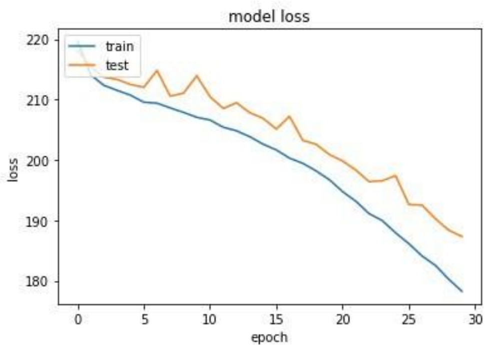


Figure 9: Pretrained CNN model – Loss Chart (30 epochs)

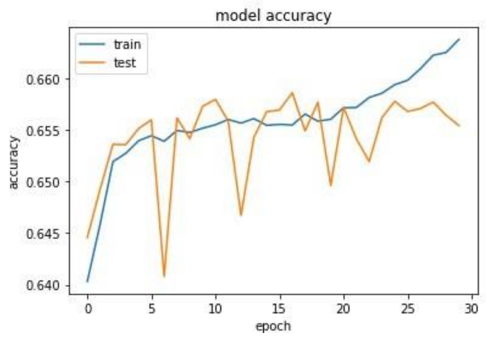


Figure 9: Pretrained CNN model – Accuracy Chart

(30 epochs)

The next illustration represents the results of the colorization of a random batch from the validationset:

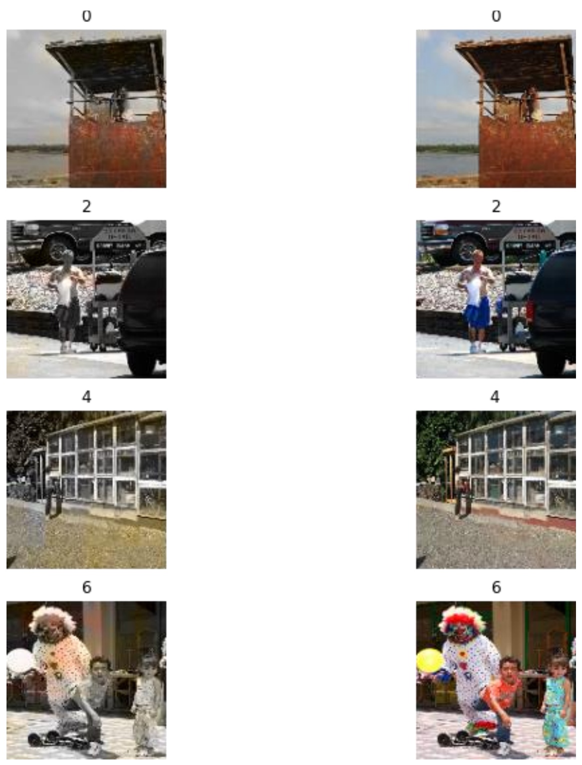


Figure 10: Original vs Colorized Images

It can be seen that the proposed CNN model has performed better than both the auto encoder model and the pretrained Xception CNN model in terms of training accuracy. The following section contains the details regarding WebApp creation using the proposed model.

8 CREATION OF WEB APP

Using the proposed CNN model a web app which accepts user input to colorize images was implemented. Creating a web app using Flask involves several steps. Firstly, the Flask web framework needs to be installed, along with any other required packages. Next, the Flask app needs to be created, and the necessary routes defined. This involves creating a Python function that is associated with a particular URL endpoint, which can be used to process requests from the user's browser. Within each route, the proposed CNN model can be called to generate predictions on the user's input. Once the predictions have been generated, they can be returned to the user's browser in the form of an HTML response. This HTML response can be generated using a template engine such as Jinja2, which allows for dynamic content to be inserted into the web page. The following screenshots illustrate the working of the Web app.

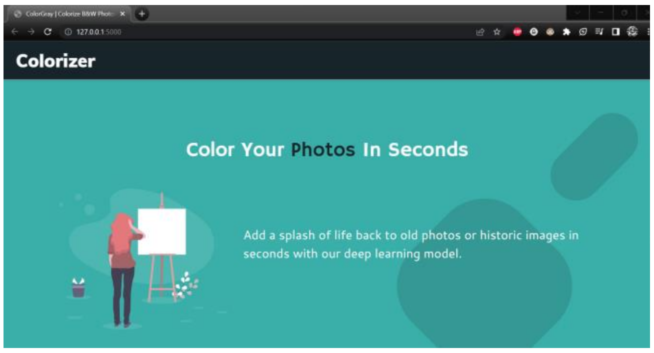

Colorizer
Color Your Photos In Seconds
Add a splash of life back to old photos or historic images in seconds with our deep learning model.

Figure 11: Web App Home Page

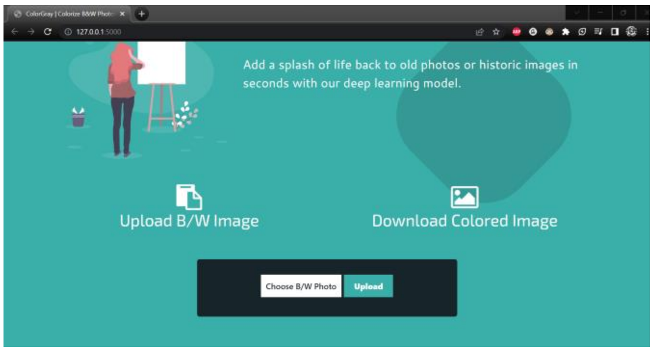

Add a splash of life back to old photos or historic images in seconds with our deep learning model.
Upload B/W Image
Download Colored Image
Choose B/W Photo
Upload

Figure 12: User uploads grayscales images

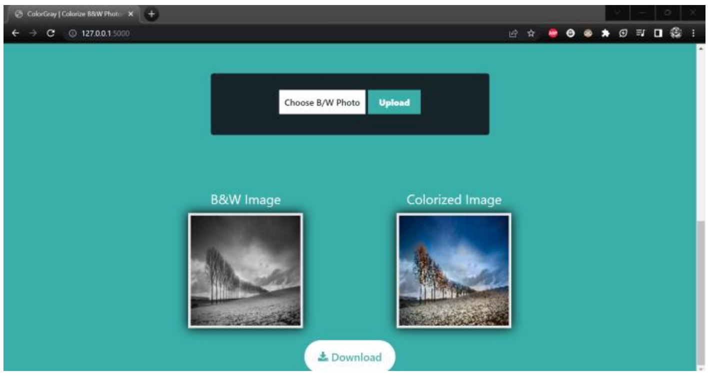


Figure 13: Colorized image is displayed

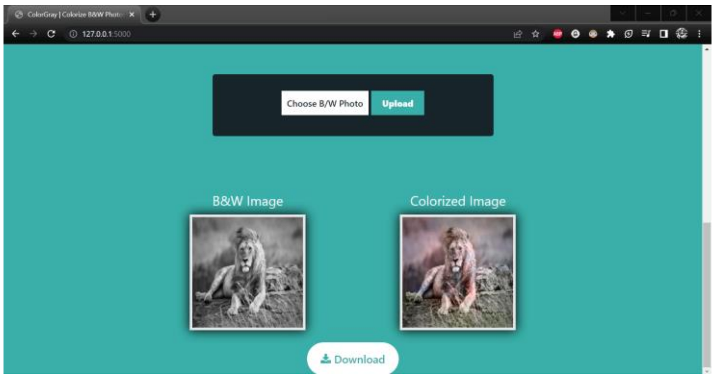


Figure 14: Colorized version of a greyscale lion image

9 CONCLUSION AND FUTURE WORKS

The problem of colorizing grayscale images has been explored using deep learning techniques. We have implemented a convolutional neural network (CNN) model and trained it on the Flickr – 8K dataset. We have achieved a validation accuracy of 65% and tested the model on various grayscale images, showing impressive results.

Furthermore, we have created a web application using Flask to provide an interactive interface for users to upload their own grayscale images and get them colorized using our CNN model. This web application has the potential to be used by photographers, graphic designers, and other professionals in the image processing industry.

Overall, this project has provided valuable insights into the application of deep learning for colorizing grayscale images and demonstrated the potential of web applications to make such technology accessible to a broader audience.

The project can be extended to coloring greyscale videos and comics in the future. A commercial colorizer can be implemented by further training the model and achieving better accuracy as well.

10 REFERENCES

[1] CIELAB color space. *Wikipedia Article: Advantages paragraph.*

[2] Flickr 30k dataset. *https://www.kaggle.com/adityajn105/flickr 30k* .

[3] Flickr 8k dataset with captions. *https://www.kaggle.com/kunalgupta2616/f lickr-8k-images-with-captions*.

[4] tf.data.dataset api. *https://www.tensorflow.org/api$_d$ocs/python /tf/data/Dataset*.

[5] Image colorization: A survey and dataset, 2020.

[6] CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. In *Proceedings ofthe IEEE conference on computer vision and pattern recognition* (2017), pp. 1251–1258.

[7] LEWINSON, E. Image colorization using convolutional autoencoders. *https://towardsdatascience.com/image-colorization-using-convolutional-autoencoders- fdabc1cb1dbe*.

[8] ZHANG, R., ISOLA, P., AND EFROS, A. A. Colorful image colorization. In *European conferenceon computer vision* (2016), Springer, pp. 649–666.