Image Colorization Using Different Deep Learning Techniques

Yash Aggarwal	Nityash Gautam	Harsh Gunwant	Parth Bhatt
862333037	862395403	862393331	862395295
yagga004@ucr.edu	ngaut006@ucr.edu	hgunw001@ucr.edu	pbhat029@ucr.edu

Abstract

Image colorization is the process of adding color to black-and-white or black-and-white images, aiming to restore and enhance their visual appeal. This project explores the application of deep learning techniques for automatic image colorization. Leveraging the power of convolutional neural networks (CNNs), Encoder-Decoder Networks and generative adversarial networks (GANs), we attempt to effectively capture the semantic context and texture details of black-and-white images to produce plausible and realistic colorization of the image.

The project begins by curating a diverse dataset consisting of black-and-white images and their corresponding color images. Various Pre-processing techniques are employed to ensure data consistency and quality. Various deep learning models then designed to learn the complex mappings between black-and-white and color images. Through an iterative training process, the model learns to generate colorized outputs that closely resemble the ground truth color images.

1 Introduction

Image colorization is an intriguing field of research that focuses on adding color to black-and-white or black-and-white images. While black-and-white images have their own charm, colorization techniques offer a way to breathe new life into these images, enhancing their visual appeal and providing a fresh perspective on historical or monochromatic visuals. In recent years, the advent of deep learning has revolutionized the field, enabling the development of advanced algorithms that can automatically colorize images with remarkable accuracy and realism.

The objective of this project is to explore the application of deep learning techniques, specifically convolutional neural networks (CNNs) and generative adversarial networks (GANs), for automatic image colorization. By harnessing the power of these neural network architectures, we aim to not only accurately predicts colors for black-and-white images but also preserves the semantic context and texture details present in the original images.

To achieve this, a diverse dataset comprising black-and-white images and their corresponding color images is curated. This dataset serves as the foundation for training the colorization model, which learns the intricate relationships between black-and-white and color images. The training process employs supervised learning methods, leveraging both pixel-level and global image-level information. Through this iterative learning process, the model becomes adept at generating colorized outputs that closely resemble the ground truth color images.

The project's contributions lie in the development of a novel Image Colorization Data-set that consists of approximately 3600 black-and-white images and their colored representations and Various Models some designed from scratch following an Encoder-Decoder design and some using Pre-existing CNN models as backbone. More information about the model designs can be found in section 4.

2 Related Work

The field of image colorization has garnered significant attention from researchers, leading to the development of various techniques and approaches. In this section, we provide an overview of some notable works that have contributed to the advancement of image colorization.

2.1 Traditional Methods

Early approaches to image colorization relied on manual intervention, where artists or users had to manually annotate black-and-white images with color information. These methods were time-consuming and subjective, often resulting in colorization that were dependent on the artist's interpretation. Later, researchers explored statistical methods, such as color histograms, Markov random fields, and texture synthesis, to infer color information based on image statistics and contextual cues. While these techniques offered some degree of automation, they were limited in their ability to produce accurate and realistic colorization.

2.2 Deep Learning Approaches

With the advent of deep learning, image colorization has witnessed a significant breakthrough. Convolutional neural networks (CNNs) have shown great promise in automatically learning the mappings between black-and-white and color images. Zhang et al. (1) introduced a pioneering work, utilizing a CNN-based architecture known as Colorful Image Colorization, which effectively captured colorization as a regression problem. The network learned to predict ab channels in the Lab color space based on the luminance channel.

Following the success of CNNs, generative adversarial networks (GANs) have been employed to improve the quality and realism of colorization outputs. Iizuka et al. (2) proposed a GAN-based model, combining a generator network for colorization and a discriminator network for adversarial training. By introducing adversarial loss, the model generated more visually pleasing and coherent colorization.

Other works have explored additional enhancements to the colorization process. Larsson et al. (3) introduced a user-guided colorization approach, allowing users to provide sparse color hints to guide the colorization process. This interactive framework leveraged deep learning techniques to propagate color information based on the user's inputs. Additionally, some researchers have explored the utilization of attention mechanisms and recurrent neural networks (RNNs) to capture long-range dependencies and improve the preservation of semantic context in colorization results.

3 Dataset

We present a dataset of images specifically curated for the task of image colorization. The dataset consists of a collection of images sourced from the popular photo-sharing platform Unsplash (4).

The initial dataset comprises a diverse range of high-resolution images in their original color format. These images were selected from various categories, encompassing landscapes, portraits, objects, people, animals and abstract compositions. The categories for images are similar to the ones provided in cifar-10 (5) and cifar-100 (6) datasets. In order to maintain consistency and facilitate colorization experiments, all images were resized to a resolution of 512x512 pixels.

To create the necessary data for image colorization, the original images were processed through a series of transformations. First, the images were converted to black-and-white using OpenCV Library (7). This step ensured that the color information was removed, simulating the input conditions for the image colorization task. The images were then resized to 512x512 pixels for uniformity.

To augment the dataset and increase its diversity, various data augmentation techniques were applied. These techniques included random rotations, vertical and horizontal flips, Gaussian blurs and random cutout. These augmentation were implemented directly in the data loaders and the images were used to create additional training examples, enriching the dataset and improving the robustness of the colorization model.

Figure 1: Image Augmentations

For each black-and-white image, a corresponding color image was generated by applying the appropriate color mapping. This mapping process involved manually assigning colors to the black-and-white pixels based on the original color images. Careful attention was paid to preserving the semantic and contextual information present in the black-and-white images, ensuring that the colorization process accurately represented the original scenes.

The final dataset consists of a paired collection of black-and-white images and their corresponding color images. The dataset is split into training, validation, and test sets, following the standard NeurIPS format. The training set comprises 80% of the total dataset, while the remaining 10% each is allocated to the validation and test sets. This distribution ensures proper training, validation, and evaluation of the image colorization models.

In summary, the dataset contains a diverse set of black-and-white images derived from original color images sourced from Unsplash. Through careful resizing, conversion to black-and-white, and application of data augmentation techniques, a comprehensive collection of black-and-white images was created. By pairing each black-and-white image with its corresponding color image, we provide a labeled dataset suitable for training and evaluating image colorization models in the context of the NeurIPS framework.

4 Models And Techniques

We are utilizing four models of diverse types. Firstly, an encoder-decoder (8) model is constructed and trained from scratch. Secondly, an encoder-decoder model with VGG-16 (9) as the encoder is employed. Thirdly, a model with a similar structure utilizing InceptionV3 (10) as the encoder is used. Lastly, an encoder-decoder model with EfficientNet (11) as the encoder is implemented. Except for the first model, the encoders are not being trained and their pre-trained weights are used.

4.1 Model 1: Encoder-Decoder Network Trained from Scratch

4.1.1 Introduction

Encoder-decoder networks are a type of deep learning architecture that function by transforming the input into a hidden or latent space using the encoder and then predicting or reconstructing the original input using the decoder based on the latent space.

In the context of image colorization, the goal is to automatically add color information to grayscale images. This task is particularly challenging as it requires the model to comprehend the semantics and context of the image in order to accurately predict suitable colors.

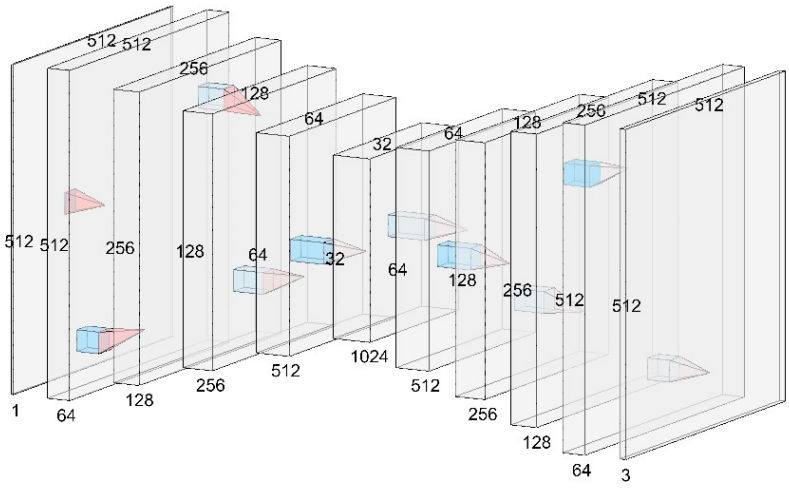
The fundamental structure of an encoder-decoder network comprises two key components: an encoder and a decoder. The encoder's role is to extract high-level features from the input grayscale image, while the decoder generates the corresponding color information. This architecture draws inspiration from the concept of auto encoders, where the encoder and decoder are trained together to reconstruct the initial input.

4.1.2 Architecture

The encoder typically consists of several convolutional layers that progressively downsample the input image, capturing hierarchical features at different levels of abstraction. This downsampling

helps to reduce the spatial dimensionality and focus on extracting more meaningful features. The decoder, on the other hand, employs transposed convolutions (also known as deconvolutions or upsampling) to reconstruct the colorized image from the encoded features.

To train the network, a large dataset of grayscale images paired with their corresponding color images is required. During training, the network learns to minimize the difference between the predicted colorized image and the ground truth color image. This is typically done using pixel-wise loss functions such as mean squared error (MSE).



source : http://alexlenail.me/NN-SVG/AlexNet.html

Figure 2: Model Architecture

The model encoder consists of convolutional layers, initially starting with a size of 512x512 pixels and 1 channel as input. It progressively reduces the spatial dimensions and increases the number of channels, eventually reaching a bottleneck or latent space of 32x32 pixels with 1024 channels. In this latent space, high-level features and semantics are encoded. Subsequently, the model begins upsampling from the bottleneck and expands back to the original size of 512x512 pixels, producing an output with 3 channels for RGB.

4.1.3 Additional Design Choices

The model encoder incorporates batch normalization layers after each convolutional layer to improve training stability. Furthermore, a dropout layer with a dropout probability of 0.5 is added following each batch normalization layer, aiding in regularizing the model.

To prevent overfitting, weight decay is applied to all layers with a decay factor of 0.001 per epoch.

The model is trained for 300 epochs using the Adam optimizer with a learning rate of 0.001. The mean squared error (MSE) loss function is utilized as the objective function during training.

The complete Model architecture can be found at onnx model and model summary

4.1.4 Generated Images and outputs

Here are some of the Generated Coloured Images as output of the encoder-decoder network.

Figure 3: Generated Images for Encoder Decoder Model

4.2 Model 2: Encoder-Decoder Network with VGG-16 Backbone

4.2.1 Introduction

VGG16 is a convolutional neural network (CNN) architecture proposed by Simonyan and Zisserman in 2014. It gained significant attention due to its simplicity and remarkable performance in image classification tasks. The name "VGG16" refers to the fact that it was developed by the Visual Geometry Group at the University of Oxford and consists of 16 weight layers.

The key feature of VGG16 is its deep architecture, characterized by a stack of convolutional layers with small receptive fields, followed by max-pooling layers. This design allows the network to extract hierarchical features from input images, capturing both low-level and high-level information. VGG16's deep structure, with multiple convolutional and fully connected layers, results in a large number of learnable parameters. This deep architecture contributes to its ability to learn complex patterns and representations, enabling high accuracy in image classification tasks.

VGG16 achieved exceptional performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014, showcasing the effectiveness of deep convolutional neural networks in computer vision. Its success has made it a popular choice for transfer learning, where pre-trained VGG16 models can be used as a starting point for various computer vision applications. Furthermore, VGG16 serves as a benchmark architecture against which other CNN models are often compared. Its simplicity and strong performance make it a valuable tool in the field of computer vision and deep learning.

4.2.2 Architecture

The implemented model consists of an autoencoder that performs colorization using the VGG16 model as a feature extractor and a separate decoder model. The VGG16 model, a pre-trained convolutional neural network (CNN) architecture, is loaded using the vgg16.VGG16() function. A new sequential model named newmodel is created to build a partial VGG16 model for feature extraction. The layers of the VGG16 model are iterated, and up to the 19th layer are added to the newmodel. All layers in the newmodel are set as non-trainable to freeze their weights during the training process. The summary of the partial VGG16 model is displayed, providing insights into the layers and their output shapes.

For feature extraction, the model takes grayscale images, converts them to RGB format, and reshapes them to match the input shape of the newmodel. The newmodel is used to extract features from the reshaped images, and these features are appended to the vggfeatures list. This feature extraction process allows capturing meaningful representations of the images using the rich hierarchical features learned by the VGG16 model.

The decoder model, named model, is then created. It comprises multiple Conv2D layers responsible for upsampling and feature reconstruction. The decoder takes the extracted VGG features and reconstructs the colorized image. The final Conv2D layer in the decoder uses a tanh activation function to output the colorized image.

To train the model, the model is compiled using the Adam optimizer, mean squared error (MSE) loss, and accuracy as the metric. Training is performed using the extracted VGG features (vggfeatures) and the corresponding ground truth color images (Y). The model is trained for 1000 epochs with a batch size of 16. By leveraging the power of the VGG16 model for feature extraction and the decoder for reconstruction, this model architecture enables effective colorization of grayscale images, leveraging the rich representation learned by the VGG16 model.

This model architecture combines the strengths of the VGG16 model, which has demonstrated exceptional performance in image classification tasks, with the flexibility of the decoder model. The VGG16 model's ability to extract hierarchical features allows the decoder to generate accurate and visually appealing colorizations. By utilizing the learned representations from the VGG16 model, the decoder effectively reconstructs colorized versions of grayscale images. The training process, utilizing VGG features and ground truth color images, further refines the model's ability to generate accurate colorizations. Overall, this approach presents a powerful framework for automatic colorization in computer vision applications.

4.2.3 Hyperparameters

HYPERPARAMETER	SELECTION
Epochs	1000
Batch Size	16
Optimizer	ADAM
Loss	Mean-Squared Error (MSE)

Table 1: SELECTED HYPERPARAMETERS

SELECTION REASONING:

- Epochs 1000: Choosing a higher number of epochs, such as 1000 in this case, allowed the model to converge to a better solution by updating the weights over a larger number of iterations. However, the choice of the number of epochs also incorporates the risk of overfitting.

- Batch Size 16: Due to the computation resources limitation, a batch size of 16 has been selected. Also, the batch size determines the number of samples processed in each training iteration. A smaller batch size, such as 16 in this case, allows for more frequent weight updates, which can result in faster convergence and better utilization of computational resources.

- ADAM Optimizer: The Adam optimizer is a popular choice for training neural networks. It combines the advantages of both AdaGrad and RMSProp optimizers, adapting the learning rate per parameter based on the first and second moments of the gradients. Adam optimizer generally performs well across a variety of tasks and offers a good balance between convergence speed and accuracy.

- MSE Loss: In colorization tasks, MSE loss provides a measure of the average difference between the colorized image and the ground truth color image.

4.2.4 Results for VGG16

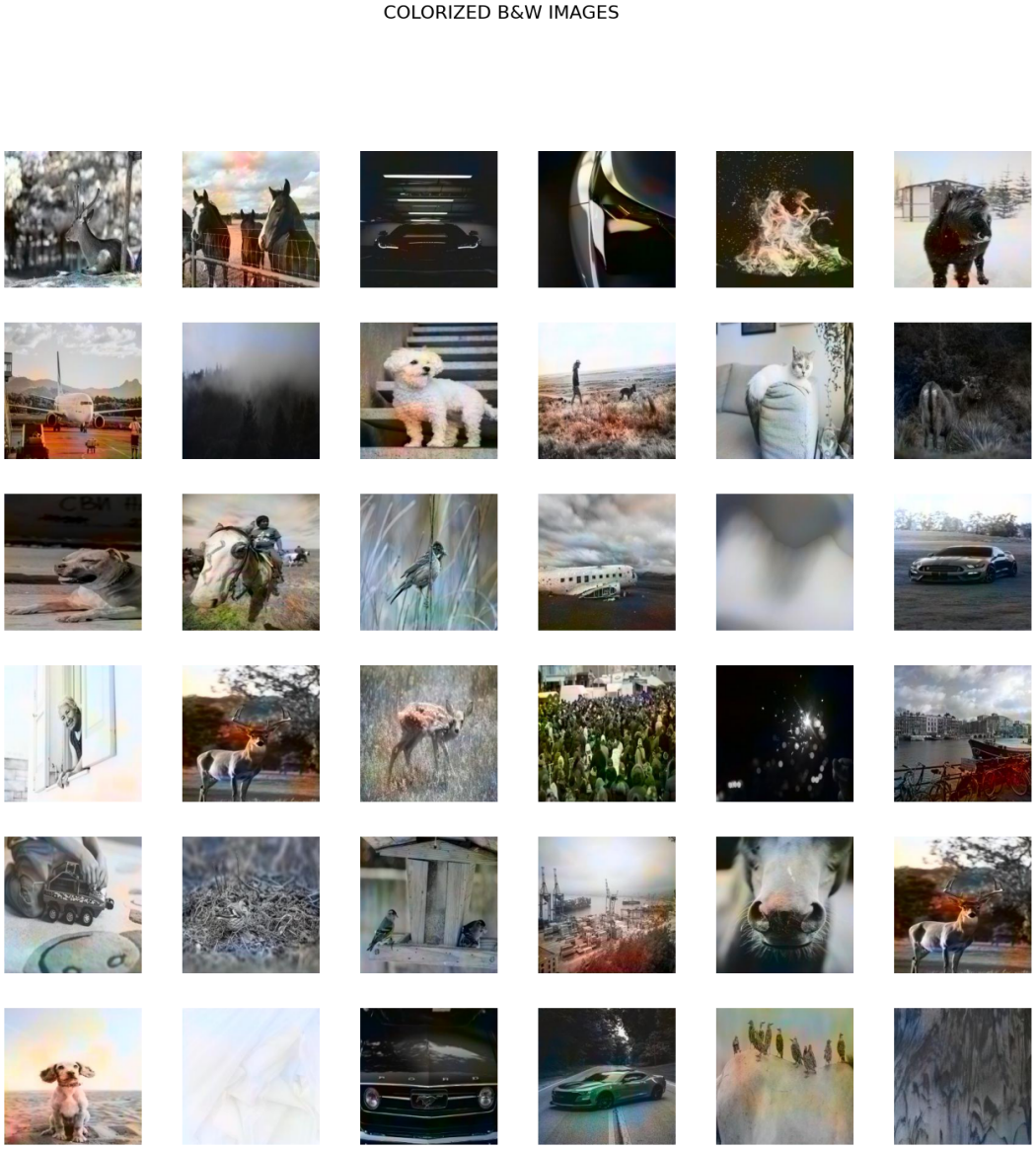


Figure 4: Colored B&W Images[1]

4.3 Model 3: Encoder-Decoder Network with Inception-V3 Backbone

Model Description and everthing to be written. Includes DEsign choices, hyperparameters and model structure and images.

4.3.1 Introduction

The Inception model, also known as GoogLeNet, is a deep convolutional neural network architecture developed by Google researchers for image classification tasks. It was introduced in 2014 and has since become a widely used model due to its impressive performance and efficiency. The Inception

[1]Click Here to Access the Image in the original code file: https://colab.research.google.com/drive/1ULzEtfBVO0e58yjX4rbTGyU9VRWHbdX8?usp=sharing

model was designed to address some of the limitations of previous models, such as the VGGNet, which were computationally expensive and had a large number of parameters.

One key feature of the Inception model is its use of the inception module, which consists of multiple parallel convolutional layers with different filter sizes. This allows the model to capture features at different scales and resolutions, enabling it to learn both local and global features effectively. By using these parallel paths, the Inception model can extract a diverse set of features in an efficient manner.

The Inception model also incorporates the idea of auxiliary classifiers, which are additional classifiers inserted at intermediate layers of the network. These auxiliary classifiers help combat the vanishing gradient problem during training and provide regularization by introducing additional supervision signals. They encourage the network to learn more discriminative features and improve gradient flow during backpropagation.

4.3.2 Architecture

The architecture of the model can be divided into three main components: the encoder, fusion, and decoder. The purpose of this model is to perform colorization of grayscale images.

The encoder component is responsible for capturing relevant features from the input grayscale images. It consists of several convolutional layers that progressively downsample the spatial dimensions while increasing the number of channels. The input to the encoder is a grayscale image of size 256x256 with a single channel. The first convolutional layer applies 64 filters of size 3x3 with a ReLU activation function. This layer helps extract low-level features from the input image. The subsequent layers continue to apply convolutions with increasing numbers of filters, allowing for the extraction of higher-level features. Strided convolutions are used at appropriate stages to downsample the spatial dimensions of the feature maps.

The fusion component combines the encoded features with an embedding vector to facilitate the colorization process. The embedding vector is obtained by passing a resized version of the grayscale image through the pretrained InceptionV3 model pre-trained. The fusion component begins by repeating the embedding vector to match the spatial dimensions of the encoded features. The repeated embedding vector is then concatenated with the encoded features along the channel dimension. This fusion of information is performed to effectively incorporate both the learned features from the encoder and the semantic information from the embedding vector. A 1x1 convolutional layer with 256 filters and a ReLU activation function is applied to fuse the features.

The decoder component aims to reconstruct the colorized image from the fused features. It consists of several convolutional and upsampling layers. The fused features from the fusion component serve as the input to the decoder. The decoder begins by applying a series of convolutional layers to progressively upsample the feature maps. These layers help in gradually increasing the spatial dimensions while reducing the number of channels. Upsampling is performed using bilinear interpolation. The final layers of the decoder apply convolutions with 2 filters and a "tanh" activation function to generate the a and b color channels of the Lab color space. The output of the decoder is then upsampled to match the original image size.

4.3.3 Hyperparameters Used

HYPERPARAMETER	SELECTION
Total Epochs	500
Steps per epoch	10
Batch Size	10
Optimizer	RMSProp
Loss	Mean-Squared Error (MSE)

Table 2: SELECTED HYPERPARAMETERS USED FOR TRAINING

4.3.4 Additional Design Choices

In order to avoid overfitting in the model, 500 epochs were chosen as it provided with good quality results. Additionally, every epoch was run for 10 steps.

The optimizer used for training the model is the RMSProp optimizer along with Mean Squared Error Loss(MSE)

4.3.5 Results

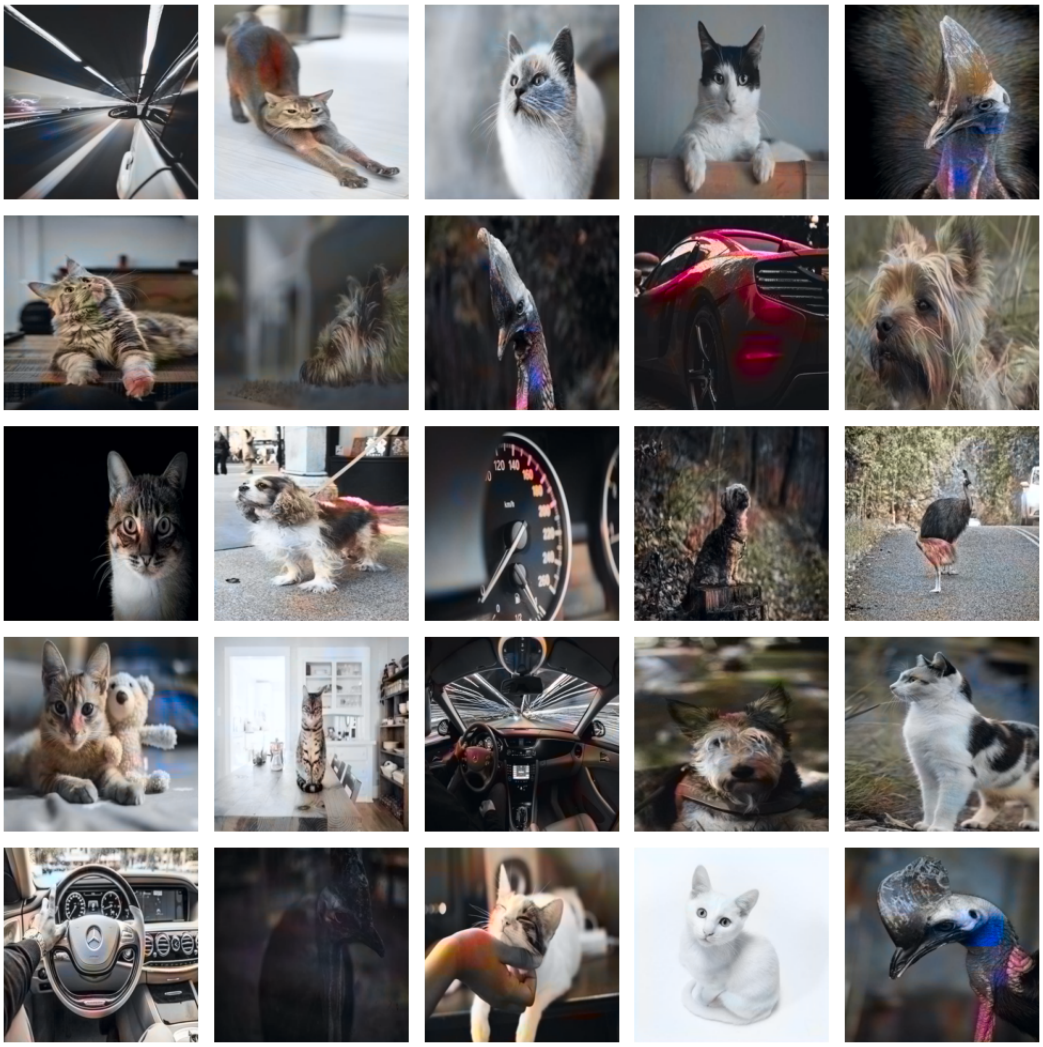


Figure 5: Results from Inception v3

The following image(figure 5) is mix of the part of the results obtained after the running the model in the test.

4.4 Model 4: Encoder-Decoder Network with EfficientNet Backbone

4.4.1 Introduction

EfficientNet is a type of Convolutional Neural Network (ConvNet) that stands out for its high efficiency, effectiveness, and accuracy [12]. Its creators introduced a novel scaling methodology for ConvNets, in which they equalize the scaling of breadth, resolution, and depth to enhance both accuracy and efficiency [11]. This approach led to the development of a series of models called

EfficientNets, consisting of eight original variants named EfficientNet-B0 through EfficientNet-B7. As we progress from B0 to B7, the models become more architecturally complex and have a greater number of parameters. Extensive testing and comparison revealed that EfficientNet-B0 achieved the best balance of accuracy and efficiency. The chosen EfficientNet models possess low Floating Point Operations Per Second (FLOPS) costs and fewer parameters, yet deliver cutting-edge performance in image categorization tasks across various datasets.

The authors of reference [11] reported the following statement: "On ImageNet, our EfficientNet-B7 achieved a top-1 accuracy of 84.4% and a top-5 accuracy of 97.1%, while being 8.4 times smaller and 6.1 times faster than the best existing ConvNet. Moreover, our EfficientNets demonstrated successful transfer learning to CIFAR-100 (91.7%), Flowers (98.8%), and three other datasets. Not only did our model surpass ResNet50 and CNN in terms of performance, but it also contained fewer parameters (5.3 million) compared to ResNet50 and CNN."

EfficientNet is a highly efficient and accurate ConvNet architecture. Its scaling methodology ensures equal consideration of breadth, resolution, and depth, resulting in effective models. The original EfficientNet variants offer a range of architectural complexities, with EfficientNet-B0 being the most favorable choice based on extensive evaluation. These models exhibit remarkable performance across various datasets, while maintaining low computational costs and parameter counts. The authors' experiments demonstrated the superiority of EfficientNet over competing models such as ResNet50 and CNN, both in terms of performance and parameter efficiency. With the aim of evaluating the performance of EfficientNets on the Image colorization that is converting gray scale images to colored images, which it had performed exceptionally well on previous datasets, we discovered that EfficientNet was an excellent choice.

4.4.2 Architecture

The architecture of EfficientNet-B0, which was designed to be both accurate and efficient in terms of Floating Point Operations Per Second (FLOPS), was created using a Neural Architecture Search (NAS) technique. The NAS technique allowed for the automatic exploration and selection of an optimal baseline architecture. Figure 3 illustrates the architecture of EfficientNet-B0. To start the

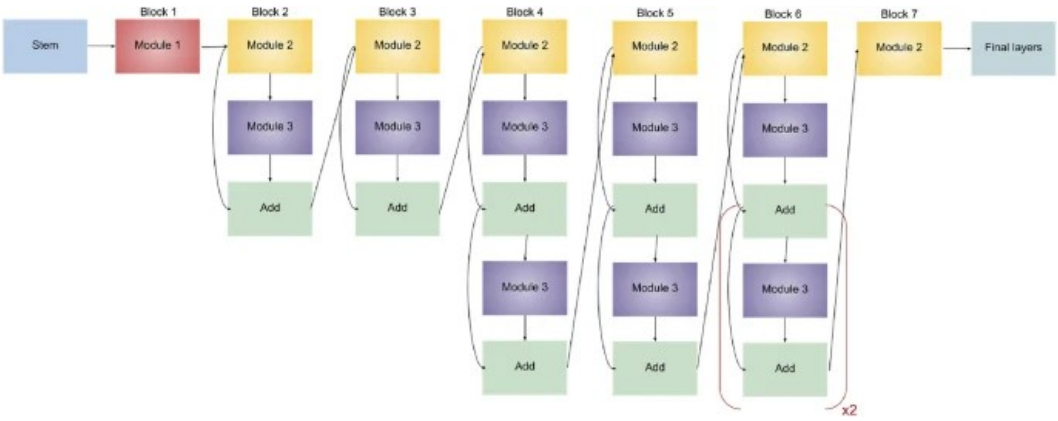


Figure 6: Model Architecture

process, the stem network is designed as the initial step. The stem network's design is depicted in Figure 4. Following this, the design process continues for the various architectures, which is a consistent approach across all EfficientNet models. Figure 5 represents the standardized design employed for these architectures.

EfficientNet models consist of seven blocks, with the number of sub-blocks varying as we progress from EfficientNet-B0 to EfficientNet-B7. EfficientNet-B0 has 237 layers, while EfficientNet-B7 has 813 layers. However, these numerous layers can be constructed using five modules that form sub-blocks, providing a more modular and scalable design. The design outlined in Figure 3 accurately describes the EfficientNet B0 model.

The five modules used in EfficientNet are as follows:

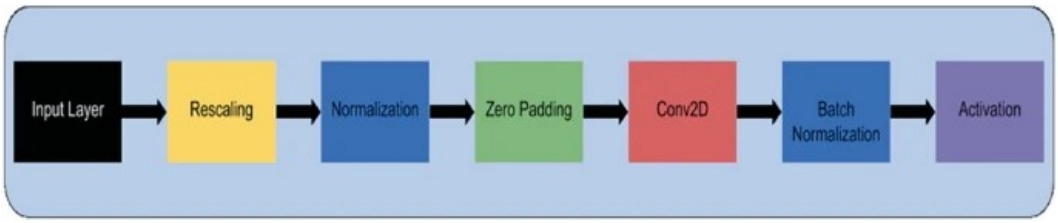


Figure 7: STEM of each architecture in EfficientNet B0-B7 Model

Figure 8: Final Layer of the architecture in EfficientNet B0 Model

(i) Module 1: The base module for sub-blocks.

(ii) Module 2: The starting point for the seven major blocks, where sub-blocks are initiated.

(iii) Module 3: An interconnected chain of sub-blocks, linked as skip connections to the main block.

(iv) Module 4: Used to connect sub-blocks, enabling each to have its skip connection.

(v) Module 5: Establishes a skip connection between the two preceding sub-blocks.

The specifics of EfficientNet-B0, such as the kernel size for convolution operations, resolution, channels, and layers, are detailed in Table 2.

Table 3: EfficientNet-B0

Stage i	Operator	Resolution	No. of Channels	No. of Layers
1	Conv3×3	224×224	32	1
2	MBConv1, k3×3	112×112	16	1
3	MBConv6, k3×3	112×112	24	2
4	MBConv6, k5×5	56×56	40	2
5	MBConv6, k3×3	28×28	80	3
6	MBConv6, k5×5	14×14	112	3
7	MBConv6, k5×5	14×14	192	4
8	MBConv6, k3×3	7×7	320	1
9	Conv1×1 & Pooling & FC	7×7	1280	1

4.4.3 Compound Scaling

Compound scaling involves scaling ConvNets in three dimensions: breadth, depth, and resolution. Previous scaling attempts focused only on one dimension. However, balancing these three dimensions leads to more efficient, effective, and accurate models. Compound scaling aims to enhance network depth for higher-resolution images, allowing for larger receptive fields that capture more pixels. Additionally, as the resolution increases, the network's breadth should be expanded to accommodate larger and more detailed patterns. It emphasizes the use of multiple scaling dimensions instead of uniformly scaling everything in a single dimension [ref2].

4.4.4 Implemented Architecture

The architecture implemented in the code combines the EfficientNetB0 model, the encoder network, the fusion module, and the decoder network to perform image colorization.

The EfficientNetB0 model, pre-trained on the ImageNet dataset, serves as a feature extractor. It takes a grayscale image as input and extracts high-level features from it. The encoder network consists

Figure 9: Implemented Architecture for EfficientNetB0

of several convolutional layers that process the grayscale image and capture important features at different levels of abstraction. These layers reduce the spatial dimensions while increasing the number of filters, allowing for the extraction of more complex features.

In the fusion module, the embeddings obtained from the EfficientNetB0 model are combined with the features extracted by the encoder network. The embeddings are reshaped and concatenated with the encoder features, and a convolutional layer is used to fuse them together. This fusion process helps incorporate the global information from the EfficientNetB0 model into the local features extracted by the encoder.

The decoder network takes the fused features as input and aims to reconstruct the colorized image. It consists of convolutional and upsampling layers that gradually upsample the features, allowing the decoder to capture finer details. The final output of the decoder network is a colorized image in the AB color channels.

Overall, the architecture leverages the pre-trained EfficientNetB0 model to extract rich features, combines them with encoder features through fusion, and uses the decoder network to reconstruct the colorized image. This enables the model to effectively perform the task of image colorization.

4.4.5 Hyperparameters Used

Table 4: Hyperparameters

Hyperparameter	**Value**
Batch Size	10
Number of Images	100
Target Image Size	(256, 256)
Encoder Filters	64, 128, 128, 256, 256, 512, 512, 256
Fusion Filters	256
Decoder Filters	128, 64, 32, 16, 2
Loss Function	MSE (Mean Squared Error)
Optimizer	RMSprop
Number of Epochs	500

In the table 4, above are the hyperparameters that the model was trained on. Below in Figure 10, is the generated images from the implemented model.

Figure 10: Results for EfficientNetB0

5 Results

Here are several color images that have been generated, along with their corresponding ground truth images and the black and white images that were utilized to produce the colorful renditions.

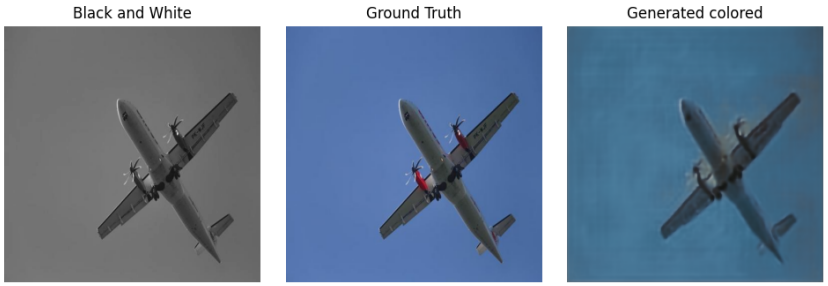


Figure 11: Generated Results for Encoder-Decoder Model

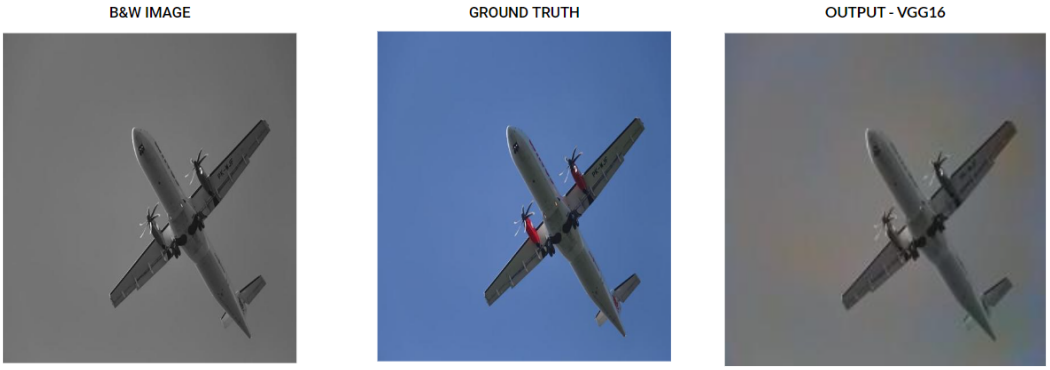


Figure 12: Generated Results for VGG16 Model

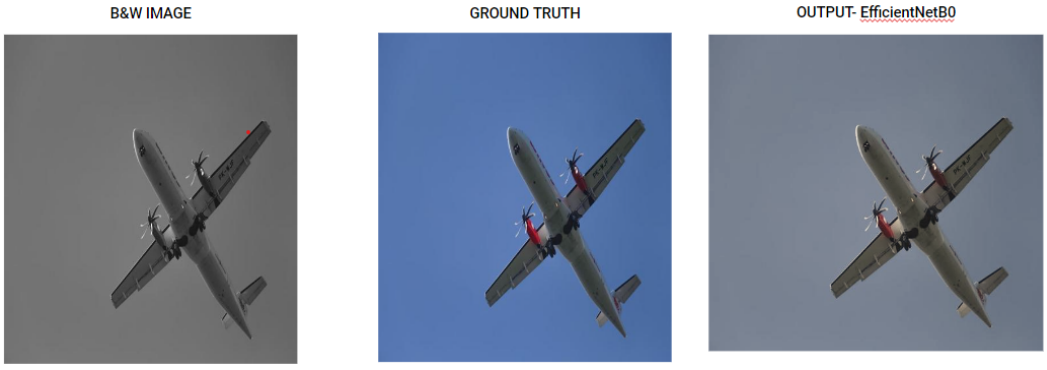


Figure 13: Generated Results for EfficientNet B0

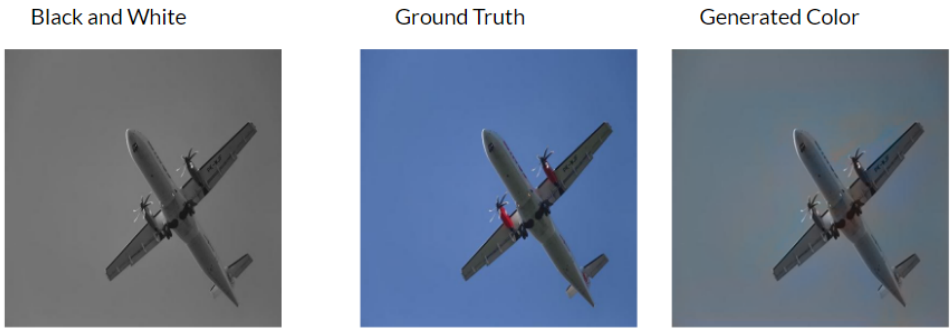


Figure 14: Generated Results for Inception V3 Model

6 Conclusion

In this Report, we presented a novel database and various approaches for image colorization using deep learning techniques. Our method leverages the power of convolutional neural networks (CNNs), Encoder-Decoder Models and generative adversarial networks (GANs) to effectively address the challenging task of adding colors to grayscale images. We also explored the influence of different network architectures, loss functions, and training strategies on the performance of our models.

7 Participation

All team members made equal contributions throughout the project. Each member actively participated in building the web scraper and diligently screened and reviewed images to ensure their suitability for the dataset. Additionally, all members played an equal role in developing the data augmentation code and implementing custom augmentations within the data loader.

Specifically, Team Member Yash took the initiative to construct the encoder-decoder architecture from scratch. Team Member B dedicated their efforts to designing the model with a VGG16 backbone, while Team Member Parth focused on implementing the Inception-V3 backbone. Team Member Harsh was responsible for creating the EfficientNet backbone. It is important to note that each member independently handled their respective models and diligently crafted the accompanying README files.

8 Links

The code for dataset generation and model 1 can be found at https://github.com/yashUcr773/CS_228_Intro_to_DL/tree/main/Projects/Project%201

The code for dataset generation and model 2 can be found at https://drive.google.com/file/d/1VmY9_wtjPeaeUbThIDiA-FI3Zxew1o7x/view?usp=sharing

The code for dataset generation and model 3 can be found at https://drive.google.com/file/d/1oB2bhpMZGC1ICArH7x2cM6BmJpgYalVN/view?usp=sharing

The code for dataset generation and model 4 can be found at https://drive.google.com/drive/folders/16qq3aWUups3BDPasgkJMUVWpcSY035WG?usp=drive_link

References

[1] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, pp. 649–666, Springer, 2016.

[2] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be color! joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification," *ACM Transactions on Graphics (ToG)*, vol. 35, no. 4, pp. 1–11, 2016.

[3] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pp. 577–593, Springer, 2016.

[4] "Unsplash," 2023. Accessed on 10 May 2023.

[5] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical Report*, 2009.

[6] A. Krizhevsky, "Cifar-100 (python version)," *Technical Report*, 2009.

[7] OpenCV Development Team, "OpenCV: Open source computer vision library," 2023. Accessed on 10 May 2023.

[8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[11] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6105–6114, 2019.