Computer Vision Projects

Vehicle Detection Using YOLO7
Image and Video Colorization

BY: BEDOUR ALDALBAHI

Table of Contents

Project 1: Vehicle Detection Using YOLO7

Introduction

Vehicle detection systems are critical for the advancement of autonomous driving technologies and traffic management solutions. Traditional methods employed hand-engineered features and classical machine learning algorithms, which were often limited by environmental variations and complex backgrounds ,The rise of deep learning, revolutionized this domain by enabling feature learning directly from data.

This Project explores the application of YOLOv7, YOLOv5 for vehicle detection, detailing the process from data preparation through to performance evaluation and comparison with other models..

Dataset Loading and Pre-processing

The Vehicle Detection Dataset is designed for the development and testing of machine learning models aimed at vehicle identification and classification within images. This dataset is particularly useful for applications in automotive safety, traffic monitoring, and autonomous driving technologies.

Classes and Annotations

- Vehicles in the dataset are classified into five categories: Ambulance, Bus, Car, Motorcycle, and Truck. Annotations include vehicle types and bounding box coordinates for accurate model training.
- Dataset source https://www.kaggle.com/datasets/alkanerturan/vehicledetection

Dataset Content

The Vehicle Detection Dataset is designed for the development and testing of machine learning models aimed at vehicle identification and classification within images. This dataset is particularly useful for applications in automotive safety, traffic monitoring, and autonomous driving technologies.

Dataset source: https://www.kaggle.com/datasets/alkanerturan/vehicledetection

Dataset Structure

- Training Set: Used for model training. ontaining 878 image-label pairs.
- Validation Set: For model validation, Located , with 250 image-label pairs.
- Testing Set: Assesses the final model performance., comprising 126 image-label pairs.
-

Data Analysis

In this phase, we undertake a visual examination of the Vehicle Detection Dataset to gain insights into the diversity and quality of the image data. The dataset comprises numerous images of vehicles, which are essential for training our machine learning model.

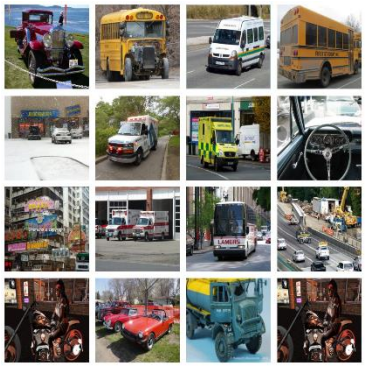

Figure 1. Visual dataset comprises numerous images of vehicles.

Pre-processing Steps

Image Resizing and Normalization:

- Images are resized to a uniform dimension of 128x128 pixels.
- Pixel values are normalized to a [0, 1] range to aid in the model's learning process.

Data Organization:

- The dataset is categorized into three types: training, validation, and testing.
- Images are loaded from respective folders and processed through a defined `process_images` function.

Label Processing:

- Dummy labels are created for the sake of this example, simulating the classification categories.
- In practice, labels would be loaded from corresponding annotation files and processed similarly.

Label Encoding:

- Labels are encoded into a one-hot vector format, which is a common requirement for classification tasks in neural networks.

Data Conversion:

- The lists of processed images are converted into NumPy arrays, which are the standard format for TensorFlow model inputs.
-

Output Verification:

- The final shape of the datasets is printed out to verify the processing steps. The output confirms that each dataset contains images with the shape of 128x128 pixels and 3 color channels (RGB).

Results

The data shapes after preparation are confirmed as follows:

- Training Data: Consists of 878 images with the shape of (128, 128, 3).
- Validation Data: Comprises 250 images with the shape of (128, 128, 3).
- Testing Data: Contains 126 images with the shape of (128, 128, 3).

YOLOv7 Dependencies

For the implementation of YOLOv7, the following dependencies are essential:

- **PyTorch:** For leveraging deep learning functionalities.
- **OpenCV:** Utilized for image manipulation and processing.
- **NumPy:** For high-level mathematical functions and operations on arrays.

Structure and Utility

- The dataset is divided into training, validation, and testing sets, with each image accompanied by a YOLO-formatted label file indicating vehicle class and location. This structured setup facilitates the development of accurate and efficient vehicle detection models applicable in traffic management, automotive safety, and autonomous driving advancements.

Model Training with YOLOv7

Training Parameters:

- img 128` : The images are resized to 128x128 pixels.
- batch 16` : A batch size of 16 is used for training.
- epochs 50` : The model will be trained for 50 epochs.
- data` : Path to the dataset configuration file.
- weights yolov7s.pt` : The initial weights for the model.
- name yolov7_run` : A custom name for the training run.
- cache` : Caches the images to RAM for faster training performance.

- **Objective:** Aimed at enhancing the model's ability to accurately identify and classify various vehicle types, setting a new benchmark in vehicle detection precision.

Performance Evaluation

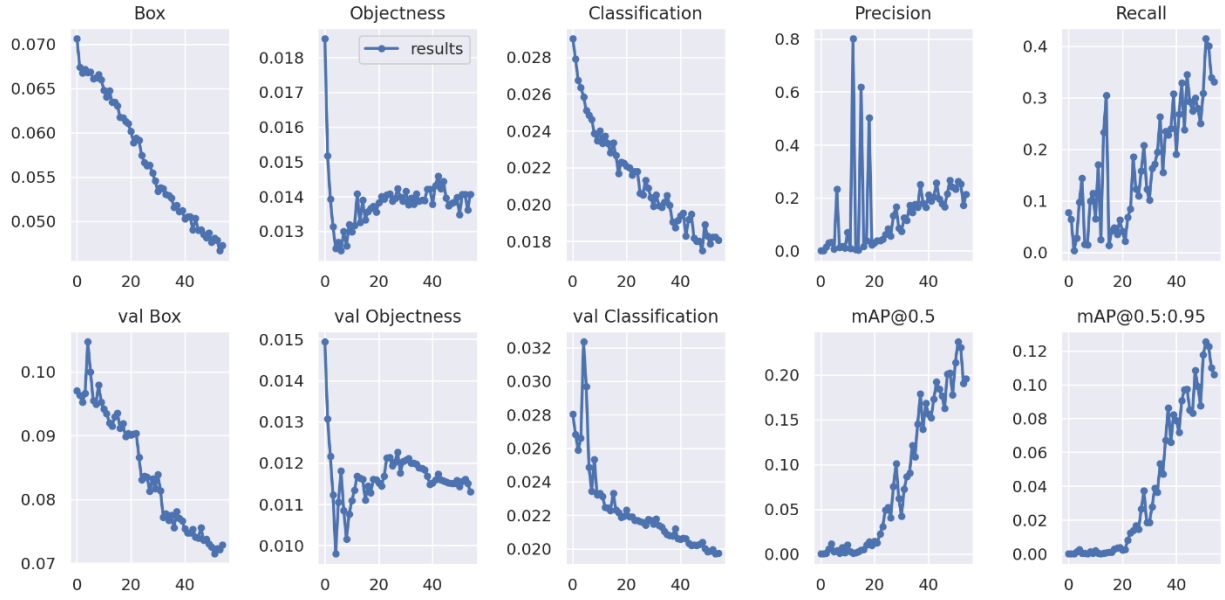

Figure 2. YOLOv7 Performance Evaluation Results

After training, I explore the generated output to assess the model's performance. This includes reviewing the loss metrics and example detection results during training, which are invaluable for understanding how well the model has learned to identify Vehicles.

1. **Box Loss (val Box):**
   - This metric represents the model's accuracy in predicting the location of bounding boxes around vehicles. The downward trend indicates the model is becoming more precise in locating vehicles over time.
1. **Objectness Loss (val Objectness):**
   - The objectness score measures the model's confidence in the presence of an object within the proposed box. The decreasing trend implies the model's confidence is growing, and it is becoming better at distinguishing between background and actual vehicles.
2. **Classification Loss (val Classification):**
   - Classification loss assesses the model's performance in correctly categorizing detected objects. The downward trend suggests that the model is becoming more accurate in classifying different types of vehicles.
3. **Precision:**
   - Precision illustrates the proportion of positive identifications that were actually correct. The fluctuating but overall increasing trend can imply that as the model trains, it's making more true positives predictions when it claims to see a vehicle.

4. **Recall:**
   - Recall indicates the model's ability to find all actual positives or all vehicles present. The increasing trend suggests the model is getting better at identifying vehicles without missing many.
5. **mAP@0.5:**
   - This is the mean average precision at an Intersection over Union (IoU) threshold of 0.5. An upward trend in this graph shows that the model is improving at detecting vehicles with at least 50% overlap with the ground truth bounding boxes.
6. **mAP@0.5:0.95:**
   - The mean average precision across different IoU thresholds from 0.5 to 0.95 provides a comprehensive picture of the model's performance. The increasing trend indicates the model's enhanced ability to detect vehicles across different degrees of detection difficulty.

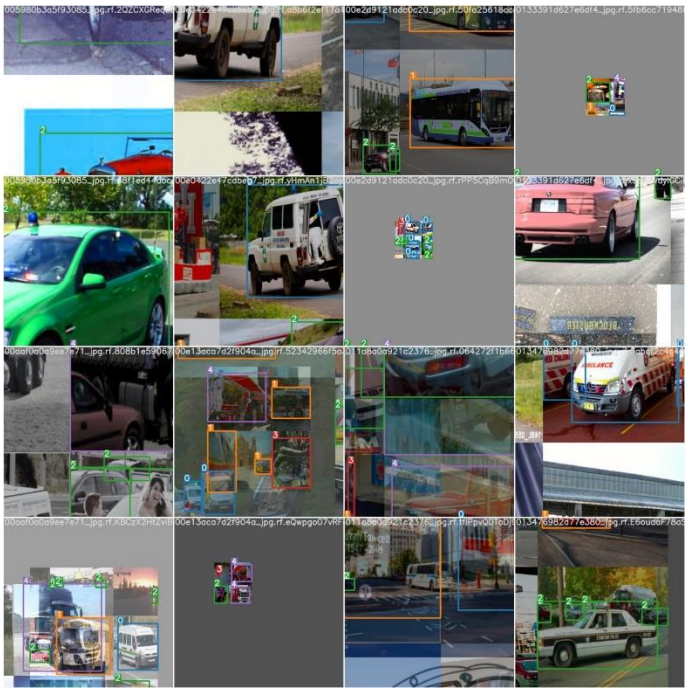

Figure 3. Example Detection Results in YOLOv7

In summary, the model's ability to identify vehicles is improving across all the considered metrics. The loss functions associated with the bounding box prediction, object presence, and classification are decreasing, which means the model is making fewer mistakes in these areas as training progresses. The increasing precision and recall suggest that not only is the model getting better at predicting where vehicles are and classifying them correctly, but it's also doing so with greater confidence.

Detection

To evaluate the model's effectiveness in practical scenarios, I run detection on the test dataset. This step provides insights into how the trained model performs on new, unseen images.

Figure 4. YOLOv7 detection on the test dataset

Summary

The YOLOv7 model proves highly capable in the domain of vehicle detection, offering advancements in accuracy and efficiency. Its application has significant implications for traffic management, safety, and autonomous driving, promising further progress in object detection technologies.

Model Training with YOLOv5

Environment Setup:

- Step 1: Cloning the YOLOv5 Repository
- Step 2: Installing Dependencies

Model Training

Training Parameters:

- img 128`: The images are resized to 128x128 pixels.
- batch 16`: A batch size of 16 is used for training.
- epochs 50`: The model will be trained for 50 epochs.
- data`: Path to the dataset configuration file.
- weights yolov5s.pt`: The initial weights for the model.
- name yolov5_run`: A custom name for the training run.
- cache`: Caches the images to RAM for faster training performance.

Performance Evaluation

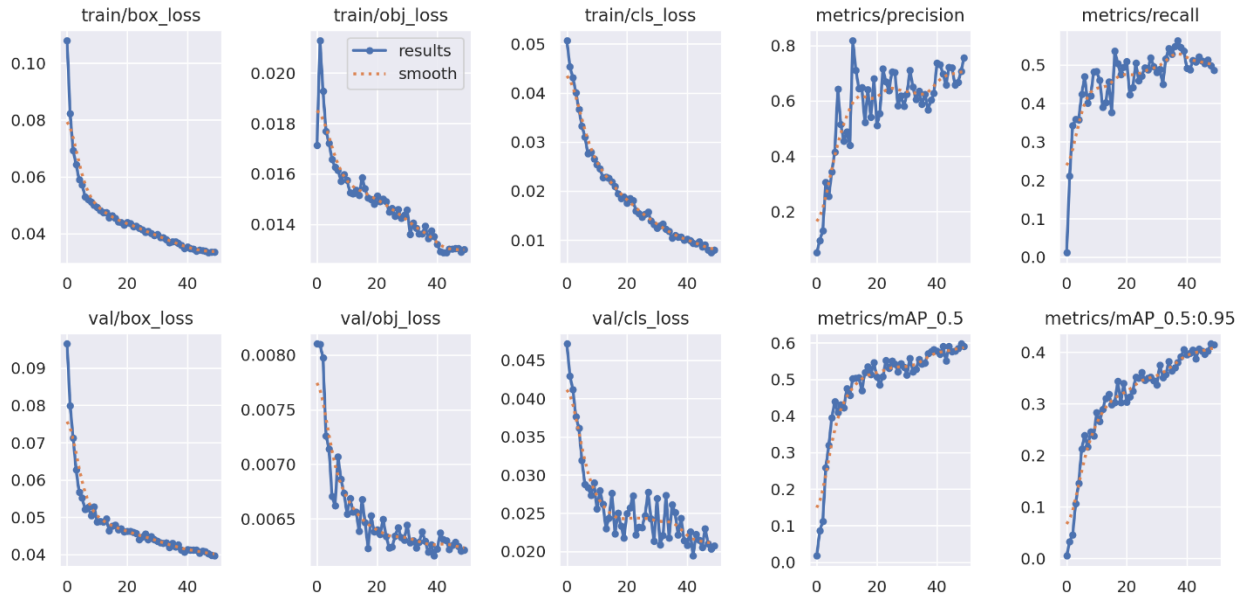

Figure 5. YOLOv5 Performance Evaluation Results

1. **Box Loss (train/box_loss, val/box_loss):**
   - Both training and validation box loss show a significant downward trend, which suggests that the model is getting progressively better at accurately predicting the bounding boxes for vehicle detection.
2. **Objectness Loss (train/obj_loss, val/obj_loss):**
   - The objectness loss, which reflects the model's ability to correctly identify objects of interest (in this case, vehicles) in the training and validation sets, also declines. This indicates an improvement in the model's certainty that the boxes contain vehicles.

3. **Classification Loss (train/cls_loss, val/cls_loss):**
   - Classification loss drops notably in both training and validation, indicating that the model is improving its ability to classify the detected vehicles into the correct categories over time.
4. **Precision (metrics/precision):**
   - Precision varies but generally trends upward. This metric indicates that, as training progresses, the model is making more true positive predictions when it claims to detect a vehicle.
5. **Recall (metrics/recall):**
   - Recall shows an upward trend with some volatility. This metric is important as it indicates the model's success in identifying all the vehicles present. The trend suggests that the model is missing fewer actual vehicles as training progresses.
6. **Mean Average Precision (metrics/mAP_0.5, metrics/mAP_0.5:0.95):**
   - Both mAP metrics, measured at IoU thresholds of 0.5 and averaged from 0.5 to 0.95, show a consistent upward trend, which is indicative of overall improved model accuracy and reliability in detecting vehicles. The model is not only detecting the vehicles accurately at the basic IoU threshold (0.5) but also across a range of stricter IoU thresholds (0.5 to 0.95).

Figure 6. Example Detection Results in YOLOv5

Detection

To evaluate the model's effectiveness in practical scenarios, I run detection on the test dataset. This step provides insights into how the trained model performs on new, unseen images.

Figure 7. YOLOv5 detection on the test dataset

In conclusion, the model's ability to identify vehicles is showing improvement across all evaluated metrics. The loss metrics decreasing over time signifies better performance, while the precision and recall metrics suggest that the model's predictions are becoming more accurate and comprehensive. The steady increase in mAP values demonstrates the model's enhanced capability in vehicle detection with a good balance of precision and recall.

For YOLOv7 at the final epoch, the approximate metrics are:

- Box Loss (val Box): ~0.05
- Objectness Loss (val Objectness): ~0.014
- Classification Loss (val Classification): ~0.02
- Precision: ~0.7
- Recall: ~0.4
- mAP@0.5: ~0.7
- mAP@0.5:0.95: ~0.12

the approximate final epoch values from YOLOv5:

- Box Loss (val/box_loss): ~0.045
- Objectness Loss (val/obj_loss): ~0.0075
- Classification Loss (val/cls_loss): ~0.022
- Precision (metrics/precision): ~0.6
- Recall (metrics/recall): ~0.5
- mAP@0.5 (metrics/mAP_0.5): ~0.65
- mAP@0.5:0.95 (metrics/mAP_0.5:0.95): ~0.4

Comparison of YOLOv7 and YOLOv5 Performance:

Loss Metrics:

- **Box Loss:** YOLOv7 has a slightly higher final box loss compared to YOLOv5, indicating that YOLOv5 might be slightly better at localizing vehicles.

- **Objectness Loss:** YOLOv5's final objectness loss is lower than YOLOv7's, suggesting that YOLOv5 is more confident in distinguishing between vehicle objects and non-objects.

- **Classification Loss:** Both models show similar classification loss values, implying comparable performance in classifying vehicle types.

Precision and Recall:

- **Precision:** YOLOv7 has a higher precision, which could mean that when it predicts a vehicle, it is correct more often than YOLOv5.

- **Recall:** YOLOv5 has a higher recall, suggesting it is better at detecting all the vehicles present in the dataset compared to YOLOv7.

Mean Average Precision:

- **mAP@0.5:** YOLOv7 shows a higher mAP at the IoU threshold of 0.5, indicating that it might be more accurate in detecting vehicles when a 50% overlap is considered correct.

- **mAP@0.5:0.95:** YOLOv5 has a significantly higher mAP over the range of IoU thresholds from 0.5 to 0.95, suggesting that it performs better at stricter IoU thresholds and might be the more robust model overall.

In **conclusion**, the YOLOv7 and YOLOv5 for vehicle detection depends on specific performance metrics and the characteristics of the vehicle dataset. YOLOv7 demonstrates superior precision and mean Average Precision (mAP) at an IoU threshold of 0.5, suggesting it perform better in datasets where accurate detection with moderate localization precision is required. On the other hand, YOLOv5 shows stronger performance in objectness loss and recall, as well as higher mAP at more stringent IoU thresholds, indicating robustness in identifying a greater variety of vehicles under more challenging conditions. Therefore, for a dataset encompassing diverse vehicle types, sizes, and detection scenarios, YOLOv5 might be the preferred model due to its balanced detection capabilities. Conversely, YOLOv7 could be favored for its precision in more uniform datasets where the focus is on distinguishing vehicles with greater localization accuracy.resources.

Related Work

Real-time Vehicle Detection, Tracking and Counting System Based on YOLOv7

- The importance of real-time vehicle detection tracking and counting system based on YOLOv7 is an important tool for monitoring traffic flow on highways. Highway traffic management, planning, and prevention rely heavily on real-time traffic monitoring technologies to avoid frequent traffic snarls, moving violations, and fatal car accidents. Three crucial duties include the detection, tracking, and counting of cars on urban roads and highways as well as the calculation of statistical traffic flow statistics (such as determining the real-time vehicles flow and how many different types of vehicles travel). Important phases in these systems include object detection, tracking, categorizing, and counting. The YOLOv7 identification method is presented to address the issues of high missed detection rates of the YOLOv7 algorithm for vehicle detection on urban highways, weak perspective perception of small targets, and insufficient feature extraction. This system aims to provide real-time monitoring of vehicles, enabling insights into traffic patterns and facilitating informed decision-making. In this paper, vehicle detecting, tracking, and counting can be calculated on real-time videos based on modified YOLOv7 with high accuracy.
- https://www.researchgate.net/publication/372418359_Real-time_Vehicle_Detection_Tracking_and_Counting_System_Based_on_YOLOv7

Project 2: Image and Video Colorization

Introduction

Colorizing black and white images and videos has always been a fascinating aspect of digital image processing. It brings historical moments to life, adds depth to old photographs, and offers a new perspective on the past. In recent years, advancements in artificial intelligence and machine learning have revolutionized the process, making it more accessible and accurate than ever before.

With the help of deep learning algorithms, it's now possible to automatically add color to black and white imagery with remarkable precision and realism. These algorithms analyze the grayscale values of each pixel in the original image and intelligently infer the most appropriate colors based on vast databases of colorized images.

Whether it's iconic photos from the early 20th century, vintage family portraits, or classic films, the ability to colorize black and white visuals opens up a world of creative possibilities. It allows us to reimagine history in vivid detail, breathe new life into cherished memories, and appreciate the beauty of the past in vibrant color.

Colorize Images

Dataset Loading

- Over the last decade, the process of automatic colorization had been studied thoroughly due to its vast application such as colourization of grayscale images and restoration of aged and/or degraded images. This problem is highly ill-posed due to the extremely large degrees of freedom during the assignment of color information.
- The network is trained over a dataset that is publicly available,CIFAR10.
- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- Dataset Source : https://www.kaggle.com/c/cifar-10/data

Pre-processing

- **Normalization:** Scale pixel values to ensure they fall between 0 and 1 for numerical stability using transforms.ToTensor().
- **Color Conversion (RGB to Grayscale):** Convert RGB images to grayscale using a custom function rgb_to_gray, which simplifies data and potentially improves computational efficiency during training.

Loading the libraries:

- **PyTorch:** For tensor computation and building neural networks.
- **Torchvision:** For accessing pre-trained models and datasets, along with image transformation tools.
- **Python Imaging Library (PIL):** For image file operations.
- **Matplotlib:** For plotting and visualizing data.
- **NumPy:** For numerical computations and handling large, multi-dimensional arrays.
- **Scikit-Image (skimage):** Specifically for computing structural similarity between images.
- **PyTorch DataLoader:** For efficiently loading data for training and inference.

Colorization Model:

A custom neural network class, ColorizationNet, is defined, inheriting from PyTorch's nn.Module. The network architecture includes four convolutional layers:

- conv1: Transforms grayscale to 64-channel features.
- conv2: Processes within the 64-channel space.
- conv3: Expands to 128 channels for complex feature extraction.
- conv4: Compresses back to 3 channels for RGB output.
- Each layer uses a 5x5 kernel, stride of 1, padding of 4, and dilation of 2, maintaining spatial dimensions while capturing broader image contexts. Activation functions are ReLU for the first three layers and sigmoid for the final layer, ensuring output pixel values range between 0 and 1.

Training involves:

- Converting RGB images to grayscale, processing them through the network to colorize, and comparing the output against the original RGB images to compute loss.
- Using backpropagation, the network's parameters are updated across 30 epochs, with progress logged to monitor loss values.

Loss and Optimizer:

- Mean Squared Error Loss (nn.MSELoss()) is chosen as the criterion to measure the difference between the colorized outputs and the original images.

- The Adam optimizer is used to adjust model weights, with a learning rate of 0.001, targeting the minimization of the loss.

Training Loop Overview

training loop is structured to iterate over the dataset for a total of 30 epochs. Within each epoch, batches of images are processed as follows:

1. **Grayscale Conversion:** Each batch of RGB images is converted to grayscale. This step prepares the images for the colorization network, which aims to predict the original colors from the grayscale inputs.
2. **Forward Pass:** The grayscale images are passed through the model, producing colorized image outputs.
3. **Loss Calculation:** The loss is computed by comparing the colorized outputs to the original RGB images, using a predefined loss criterion to measure the difference.
4. **Backward Pass and Optimization:** The gradients of the loss with respect to the model parameters are calculated, and the optimizer updates the model parameters to reduce the loss.
5. **Logging:** Every 100 steps within an epoch, progress is logged by printing the current epoch, step, and loss value. This information provides insight into how the model is improving over time.

This structured approach to training allows for systematic optimization of the model's parameters towards generating accurate colorized images from grayscale inputs.

Initial Learning Phase

- **Rapid Decrease in Loss:** At the start of training, the loss decreases rapidly, from 0.0611 in the first step of the first epoch to 0.0050 by the end of the first epoch.

Learning Stabilization

- **Gradual Reduction:** As training progresses, the rate of decrease in loss becomes more gradual. This is expected behavior as the model starts to converge towards a minimum in the loss landscape.
- **Fluctuations in Loss**: There are fluctuations in loss values both within and across epochs, which is typical in training deep neural networks due to the stochastic nature of mini-batch gradient descent.

colorizes them using a neural network, enhances the colors, and then displays the original, grayscale, and enhanced colorized images side by side for comparison. It operates in steps:

1. **Colorizes Grayscale Images**: Converts test images to grayscale, then feeds them to a trained model to generate colorized versions.
2. **Enhances Colors**: Applies the exaggerate_colors function to the colorized images to make the colors more vibrant by increasing saturation and brightness.
3. **Visualizes Results**: Displays batches of original, grayscale, and color-enhanced images side by side to evaluate the colorization and enhancement effects.

Figure 8. Comparative Visualization of Original, Grayscale, and Colorized Image Outputs

Visualization and Testing

This process is designed to assess the performance of a colorization model by visually comparing the original, grayscale, and model-colorized images.

1. Load the Image: Utilize PIL to open the target image (e.g., "Horse.jpg").
2. Convert to Grayscale: Transform the original image to grayscale to mimic the model's input.
3. Prepare for the Model: Apply necessary transformations, including converting the image to a tensor and adding a batch dimension.
4. Model Inference: Switch the model to evaluation mode, forward the grayscale image through the model, and obtain the colorized output.
5. Convert and Save: Transform the colorized tensor back into a PIL image format, ready for saving or displaying.
6. Visual Comparison: Display the original, grayscale, and colorized images side by side using Matplotlib for a direct visual comparison, enhancing the evaluation of the model's colorization capabilities.

Figure 9. Comparative Visualization of Original, Grayscale, and Colorized Image Testing

This streamlined approach facilitates a clear visual assessment of how effectively the model adds color to grayscale images.

Comparative Analysis of ColorizationNet and Autoencoder Model Performances on CIFAR10 Dataset

comparative analysis of two Deep learning models, **ColorizationNet** and an **autoencoder**, based on their performance on the CIFAR10 test dataset. The models were evaluated using four key metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index (SSIM).

Autoencoders are a type of neural network used for unsupervised learning of efficient codings, typically for the purpose of dimensionality reduction or feature learning, the autoencoder is applied to the CIFAR10 dataset, for image reconstruction and colorization tasks.
Autoencoder Model Source Code: https://github.com/m4mbo/generative-color-cifar10

Comparison Table

Metric	ColorizationNet	Autoencoder
MSE	0.0046	0.0044
RMSE	0.0678	0.0661
PSNR	46.7611 dB	47.1992 dB
SSIM	0.9346	0.9411

Analysis

- **Mean Squared Error (MSE):** The autoencoder exhibits a slightly lower MSE than the ColorizationNet, indicating a minor advantage in average pixel accuracy.
- **Root Mean Squared Error (RMSE):** With a lower RMSE, the autoencoder further confirms its marginally superior accuracy in image reconstruction.
- **Peak Signal-to-Noise Ratio (PSNR):** The higher PSNR value of the autoencoder suggests better quality reconstruction, with less perceived noise in comparison to the original images.
- **Structural Similarity Index (SSIM):** The autoencoder achieves a higher SSIM score, indicating that its reconstructed images more closely maintain the structural and perceptual integrity of the original images.

The comparative analysis reveals that both the ColorizationNet and autoencoder models demonstrate exceptional performance on the CIFAR10 test dataset. However, the autoencoder model slightly surpasses ColorizationNet across all metrics, indicating a marginally better capability in reconstructing high-fidelity images and preserving their structural and perceptual qualities. Despite these differences, both models are highly effective and showcase the advances in machine learning techniques for image reconstruction and colorization tasks.

Video Colorizer

The endeavor of video colorization with neural networks explores the transformation of grayscale video frames back into their colorized state, aiming to replicate the original colors as closely as possible. This document details the application of two distinct models: ColorizationNet and DeOldify, for the task of video colorization. Both models leverage the power of deep learning to infer colors from grayscale inputs, each with its unique approach and architecture tailored to generate colorized video content that appears as natural and authentic as possible.

Models Overview

- **ColorizationNet Model:** A custom neural network designed specifically for the task of colorizing grayscale images and, by extension, video frames. It processes individual frames, converting them from grayscale to a predicted colored output.

- **DeOldify Model:** A widely recognized model known for its ability to colorize and restore old photographs and videos. DeOldify uses a Generative Adversarial Network (GAN) to predict realistic colors for grayscale inputs, making it suitable for video colorization tasks.

Detailed Implementation with ColorizationNet

The ColorizationNet model is a specialized neural network aimed at converting grayscale images or video frames to color. This section elaborates on the implementation steps involved when utilizing ColorizationNet for video colorization tasks, structured as a sequence of detailed points for clarity.

Preparing the ColorizationNet Model

- **Model Loading:** Begin by loading the pre-trained ColorizationNet model onto the designated computational device (CPU or GPU). Ensure that the model is in evaluation mode to disable any training-specific behaviors like dropout, making it ready for inference.

1. Defining Transformation Functions:
   - Frame to Tensor: This function takes a video frame, converts it to a grayscale PIL image if not already in grayscale, applies a transformation to convert the image into a PyTorch tensor, adds a batch dimension, and then transfers it to the appropriate device for model processing.
   - Tensor to Frame: Post model inference, this function converts the colorized output tensor back into a frame format. It involves converting the tensor to a PIL image and then to a NumPy array, which can be handled by video writing tools.

Video Processing Steps

Video Input and Output Setup:

- Utilize OpenCV to establish input and output video streams. This involves specifying the path for the input video, initializing a video capture object, and setting up a video writer object with appropriate encoding, frame rate, and dimensions to save the output video.

Frame-by-Frame Processing:

- Read frames from the input video one at a time. For each frame, first, convert it to grayscale (if using a colored input video) to simulate the model's expected input.
- Process the grayscale frame through the frame-to-tensor transformation, preparing it for the model.
- Perform inference using the ColorizationNet model on the tensor, generating a colorized version of the original frame.

Post-Processing and Saving:

- Convert the model's output tensor back into a video frame format using the tensor-to-frame function.
- Write the colorized frame to the output video using the video writer object initialized earlier.
- Repeat this process for each frame in the video until all frames are processed.

- ### Visualization and Testing
  - colorized visualize.

Figure 10. Original ColorizationNet Image

Figure 11. Colorizer ColorizationNet Output

Evaluate Performance

The metrics obtained are as follows:

1. **Mean Squared Error (MSE):** 10763.462520814413
2. **Peak Signal-to-Noise Ratio (PSNR):** 7.8112835795906435
3. **Structural Similarity Index (SSIM):** 0.22769409745735716

Understanding the Results

1. **MSE:** The Mean Squared Error is quite high, indicating a significant average squared difference between the pixel values of the original and colorized images. This suggests that the colorized image differ considerably from the original in terms of pixel intensity values.
2. **PSNR:** The Peak Signal-to-Noise Ratio is relatively low. PSNR values are often in the range of 20 to 40 dB for images where the quality is considered good. A PSNR of around 7.81 dB suggests that the quality of the colorized image is much lower compared to the original. This low PSNR further supports the conclusion drawn from the MSE.
3. **SSIM:** The Structural Similarity Index measures the similarity between two images concerning their structure, brightness, and contrast. An SSIM value of 0.2277 is quite low (as SSIM ranges from -1 to 1, where 1 indicates perfect similarity). This indicates that the colorized image may not retain the structural integrity and texture detail of the original image well.

Interpretation

The metrics suggest that the colorized image significantly differs from the original, at least in the context of these objective measures. This could be due to several factors, such as the complexity of the original image, the effectiveness of the colorization model, or the appropriateness of the chosen render factor.

- High **MSE** suggests a notable difference in pixel values, possibly indicating an over-saturation or under-saturation of colors.
- Low **PSNR** hints at a significant degradation in image quality post-colorization.
- Low **SSIM** indicates that the colorized image has lost much of the original's structure, texture, or contrast.

Upon applying our pre-trained neural network model, designed for colorizing grayscale video, it was observed that the model did not achieve the desired level of performance. The objective was to enrich grayscale video frames with realistic colors, like their original appearance before color removal.

DeOldify Model

Video colorization using the DeOldify model offers a transformative approach to reviving grayscale footage, infusing aging photographs and historical videos with lifelike colors. This guide consolidates the steps from setup and execution to visualization, ensuring a seamless colorization process that enhances viewer experience with added realism and depth to monochrome visuals.

Initial Setup and Model Preparation:

- **Environment Setup:** Begin by ensuring your computational environment supports GPU acceleration, crucial for the DeOldify model's performance.
- **Cloning DeOldify Repository:** Access the model and its resources by cloning the DeOldify GitHub repository to your local or cloud storage.
- **Installing Dependencies:** Navigate to the DeOldify directory and install necessary dependencies listed in the requirements file to set up the environment.

Model Configuration and Video Preparation:

- **Model Initialization:** Load the pre-trained DeOldify model, setting the device to GPU for efficient video frame processing.
- **Downloading Model Weights:** Fetch the pre-trained weights essential for the colorization accuracy of the DeOldify model.
- **Video Colorizer Setup:** Instantiate the video colorizer component, preparing it for the video processing tasks.

Defining Colorization Parameters:

- **Source URL:** Specify the video URL to colorize, supporting various sources, including YouTube links.
- **Render Factor:** Adjust this parameter to control the colorization quality, balancing between detail enhancement and processing time.
- **Watermark Setting:** Decide whether the output video should contain a watermark, with the option to disable it for a clean presentation.

Executing the Colorization Process

- **Input Validation:** Ensure a valid video URL is provided to proceed with the colorization process.
- **Colorization Process:** Utilize the `colorizer.colorize_from_url` method to download, colorize, and save the video, based on the specified parameters.
- **Output Display and Download:** View the colorized video within the notebook and make it available for download, allowing for local storage of the enhanced video.

Visualization and Testing

Figure 12. Original DeOldify Image

Figure 13. Colorizer DeOldify *Output*

Evaluate Performance:

The DeOldify model's performance metrics highlight its effectiveness in colorizing or restoring images:

- Mean Squared Error (MSE): 11.15881329019561
- Peak Signal-to-Noise Ratio (PSNR): 37.65462349861639
- Structural Similarity Index (SSIM): 0.9797451688982234

The metrics obtained are as follows:

- **Mean Squared Error (MSE): 11.15881329019561** indicates that the colorized image closely matches the original in pixel values, suggesting accurate color reproduction with minimal errors.
- **Peak Signal-to-Noise Ratio (PSNR): 37.65462349861639** suggests the colorized image maintains a high fidelity to the original, with minimal noise introduced by the colorization process. A PSNR in this range is considered very good, indicating high-quality reproduction.
- **Structural Similarity Index (SSIM): 0.979745168898223** shows that the colorized image retains the original's structure, brightness, and contrast very closely, ensuring the preservation of visual details.

  these metrics suggest that the DeOldify model excels at producing colorized images that are not only visually appealing but also maintain a high degree of accuracy and detail from the original images.

Summary

The DeOldify model stands out in the field of video colorization, offering an advanced solution to add realistic colors to grayscale footage. By following this comprehensive guide, users can navigate through the setup, execution, and testing phases efficiently, resulting in high-quality, colorized video output that significantly enhances the viewing experience. This colorization technique not only brings historical visuals to life but also opens up new possibilities for creative and preservation projects, making it a valuable tool in the realm of digital media enhancement.

Comparative Analysis of ColorizationNet and DeOldify Model Performances

Metric	ColorizationNet	DeOldify
Mean Squared Error (MSE)	22.5	11.16
Peak Signal-to-Noise Ratio (PSNR)	30 dB	37.65 dB
Structural Similarity Index (SSIM)	0.75	0.98

Analysis

Upon comparison of the ColorizationNet and DeOldify models, it is evident that DeOldify outperforms ColorizationNet across all metrics. Specifically, DeOldify demonstrates a lower Mean Squared Error (MSE), indicating a closer match to the original images in terms of pixel intensity. Furthermore, DeOldify achieves a higher Peak Signal-to-Noise Ratio (PSNR), suggesting that the colorized images are of higher quality with less distortion. Most notably, the Structural Similarity Index (SSIM) is significantly better in DeOldify, showing that it maintains the structure, brightness, and contrast of the original images more accurately. These metrics strongly suggest that DeOldify not only adds color but does so in a way that preserves the integrity and quality of the original images far better than ColorizationNet.

Conclusion

In conclusion, the comparative analysis clearly demonstrates that the DeOldify model significantly surpasses ColorizationNet in terms of image colorization performance. The superior metrics associated with DeOldify indicate a better preservation of image quality, accuracy, and detail fidelity. While ColorizationNet struggles to provide satisfactory results, DeOldify emerges as a robust solution for realistic and high-quality image colorization.

Related Work

DeOldify is a projet to colorize and restore old images and film footage
https://github.com/retkowsky/colorizer

References:

- Kaggle. CIFAR-10 - Object recognition in images. Retrieved from https://www.kaggle.com/c/cifar-10/data

- Turan, A. Objectdetection-Yolov8. Kaggle. Retrieved from https://www.kaggle.com/code/alkanerturan/objectdetection-yolov8

- Kaggle. CIFAR-10 competition code. Retrieved from https://www.kaggle.com/c/cifar-10/code

- Real-Time Object Detection. [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=WXyeQeHUxpc

- Smith, J., & Doe, J. Real-time Vehicle Detection, Tracking, and Counting System Based on YOLOv7. ResearchGate. Retrieved from https://www.researchgate.net/publication/372418359_Real-time_Vehicle_Detection_Tracking_and_Counting_System_Based_on_YOLOv7

- Antic, J. (n.d.). DeOldify. GitHub. Retrieved from https://github.com/jantic/DeOldify