A red star with red text

Description automatically generated

shrog ahmad ALDALBAHI

Computer Vision Project: Deep Learning Strategies for Face Mask Detection and Colorization of Grayscale Images and Videos"

Computer Vision

Contents

[Face Mask Detection 3](#_Toc162869102)

[1. Introduction 3](#_Toc162869103)

[1.2 Related Work 3](#_Toc162869104)

[2. Importing the Dependencies 4](#_Toc162869105)

[3. Dataset Loading 5](#_Toc162869106)

[4. Data Pre-processing 6](#_Toc162869107)

[5. YOLOv7 8](#_Toc162869108)

[5.1 Creating Dataset for YOLOv7 8](#_Toc162869109)

[5.2 Dataset Structure Organization 8](#_Toc162869110)

[5.3 Dataset Partitioning 8](#_Toc162869111)

[5.4 Configuration File Generation 9](#_Toc162869112)

[5.5 Downloading YOLOv7 9](#_Toc162869113)

[5.6 Implementation 9](#_Toc162869114)

[6. Training YOLOv7 10](#_Toc162869115)

[6.1 Training Execution 10](#_Toc162869116)

[6.2 Training Output Details 10](#_Toc162869117)

[7. Model Performance Metrics 11](#_Toc162869118)

[8. Detection Evaluation 12](#_Toc162869119)

[9. Training and Results of YOLOv5 14](#_Toc162869120)

[10. Comparative Analysis of YOLO Models 15](#_Toc162869121)

[10.1 YOLOv5 vs YOLOv7 16](#_Toc162869122)

[10.2 Insight into Previous YOLO Versions 17](#_Toc162869123)

[11. Conclusion 17](#_Toc162869124)

[Modern Computer Vision Techniques to Colorize Black and White Images and Videos 19](#_Toc162869125)

[Image colorization 19](#_Toc162869126)

[1. Introduction 19](#_Toc162869127)

[2. Methodology 20](#_Toc162869128)

[3. Data Preprocessing 20](#_Toc162869129)

[4. Model Architecture 22](#_Toc162869130)

[5. Training the Model 22](#_Toc162869131)

[6. Evaluating the Model's Performance 23](#_Toc162869132)

[7. Visualizing Colorization Results 23](#_Toc162869133)

[8. Comparison of Model Performance with State-of-the-Art Models 24](#_Toc162869134)

[9. Conclusion 26](#_Toc162869135)

[Videos colorization 27](#_Toc162869136)

[1. Introduction 27](#_Toc162869137)

[2. Methodology 28](#_Toc162869138)

[3. Data Preparation 28](#_Toc162869139)

[4. Model Architecture 28](#_Toc162869140)

[5. Training Process 28](#_Toc162869141)

[6. Evaluation of Model & Performance Metrics 29](#_Toc162869142)

[7. Results 30](#_Toc162869143)

[8. Comparing Models: Autoencoder vs. DeOldify 31](#_Toc162869144)

[8.3 Comparison and Analysis 32](#_Toc162869145)

[9. Conclusions from the Model Comparison 33](#_Toc162869146)

[10.references 34](#_Toc162869147)

# **Face Mask Detection**

# 1. Introduction

In the wake of the global health crisis triggered by COVID-19, the importance of preventive measures such as wearing face masks cannot be overstated. Face masks have been identified as a simple, yet effective tool in reducing the transmission of the virus, particularly in public settings where social distancing may be challenging to maintain. In this context, the ability to automatically detect the presence and proper use of face masks using computer vision technologies has emerged as a crucial area of research and application. This project explores the application of cutting-edge object detection models, specifically YOLOv5 and YOLOv7, to the task of face mask detection. By leveraging deep learning, the project aims to accurately identify individuals wearing masks, those not wearing masks, and masks worn incorrectly in real-time, providing an invaluable tool for enhancing public health and safety measures.

## 1.2 Related Work

The application of deep learning and object detection models for public health surveillance, particularly face mask detection, has seen significant interest in recent years. This surge can be attributed to the COVID-19 pandemic, which underscored the need for automated systems to monitor compliance with health guidelines in public spaces. The use of Convolutional Neural Networks (CNNs) has been pivotal in this research area, given their proficiency in handling image data and recognizing complex patterns.

**Early Efforts:** Initial attempts at face mask detection often relied on traditional image processing techniques combined with machine learning classifiers such as Support Vector Machines (SVMs) and k-Nearest Neighbors (k-NN). However, these methods struggled with variability in lighting, occlusion, and complex backgrounds, leading researchers to explore more robust solutions.

**YOLO Series:** The introduction of the You Only Look Once (YOLO) framework marked a significant advancement in object detection, offering a fast and efficient alternative to region proposal-based methods like R-CNN. Subsequent iterations, including YOLOv3, YOLOv4, and the latest, YOLOv5 and YOLOv7, have introduced improvements in speed, accuracy, and model complexity. These models have been adopted for a wide range of applications, from pedestrian detection to wildlife monitoring, and, pertinent to this project, face mask detection.

**Comparative Studies:** Several studies have compared the performance of different YOLO versions and other object detection models in the context of face mask detection. For instance, research has demonstrated the superior accuracy of YOLOv4 over earlier versions, attributing this to its optimized architecture and enhanced feature extraction capabilities. However, there is still a need for comprehensive comparisons that include the very latest versions, such as YOLOv7, to inform best practices in the development of face mask detection systems.

# 2. Importing the Dependencies

To undertake the task of face mask detection using the YOLOv5 and YOLOv7 models, a suite of programming libraries and modules is essential. These dependencies form the foundation for handling a wide range of tasks, including data manipulation, image processing, and the visualization of results. Below, we outline the key libraries imported for this project and their respective roles:

* **NumPy (numpy):** A fundamental package for scientific computing in Python. It provides support for efficient operations on multi-dimensional arrays and matrices, which are crucial for manipulating image data and annotations.
* **Pandas (pandas):** An open-source data analysis and manipulation tool, Pandas is instrumental in handling structured data. While its direct application in this project may be limited, it's invaluable for exploratory data analysis and the management of annotation files in tabular formats.
* **Matplotlib (matplotlib.pyplot):** A comprehensive library for creating static, interactive, and animated visualizations in Python. Matplotlib is used extensively throughout this project to visualize data and results, including the display of images with bounding boxes to verify the accuracy of mask detection.
* **ZipFile (from zipfile):** This module is used for reading and writing ZIP archive files. It's particularly important in the initial stages of the project for extracting the dataset, which is often downloaded in compressed formats to reduce storage and transfer times.
* **XML Processing (xml.etree.cElementTree as ET):** The ElementTree XML API simplifies the processing of XML files. In the context of this project, XML files contain annotations for the images in the dataset, detailing the coordinates of bounding boxes and class labels (e.g., with\_mask, without\_mask). Parsing these files correctly is essential for preparing the data for model training.
* **Glob (glob):** This module is used to retrieve files/pathnames matching a specified pattern. It is crucial for automating the processing of image and annotation files, enabling efficient batch operations.
* **OS (os):** Provides a portable way of using operating system-dependent functionality. In this project, it's used for file operations like creating directories, and path manipulations, ensuring the project's directory structure is correctly set up.
* **JSON (json):** Used for parsing and manipulating JSON files. While its use in this specific script snippet is not directly mentioned, JSON plays a key role in configuration files and the serialization of data structures.
* **Random (random):** This module implements pseudo-random number generators for various distributions. Random operations can be essential for data shuffling or generating random subsets of data for validation purposes.
* **Shutil (shutil):** Offers high-level operations on files and collections of files. In this project, it's particularly useful for copying annotation and image files into appropriate directories during the data preparation phase.
* **Pillow (from PIL):** The Python Imaging Library (PIL), through its fork Pillow, adds image processing capabilities to your Python interpreter. This library is used for tasks such as reading images, applying transformations like converting images to grayscale, and augmenting the dataset to improve model robustness.

By importing these dependencies, we ensure that all the necessary tools are available for the various stages of the face mask detection project.

# 3. Dataset Loading

At this stage, we are tasked with loading and preprocessing the dataset required for my face mask detection project. The chosen dataset, available on Kaggle, contains images annotated to indicate whether individuals are wearing masks, not wearing masks, or wearing masks incorrectly. To begin processing this dataset, I first configure my environment to interact with the Kaggle API, enabling me to download the dataset directly.

The dataset is a crucial component of my project, providing the raw images and annotations needed to train and evaluate the object detection models. Here are the steps I follow to set up my environment and access the dataset:

**[Dataset source](https://www.kaggle.com/datasets/andrewmvd/face-mask-detection/code" \t "_new)**

This dataset forms the foundation of my project, providing the images that I will use to train and evaluate the YOLOv5 and YOLOv7 models for the task of face mask detection.

# 4. Data Pre-processing

The initial phase of this project involves meticulous preparation of the dataset to facilitate the application of YOLOv7 for face mask detection. A pivotal element of this preparation is the conversion of dataset annotations from the XML format, commonly employed in datasets like Pascal VOC, to the TXT format necessitated by the YOLO architecture. This conversion is indispensable for enabling YOLO to accurately interpret object locations and classes within the images.

**Handling XML Annotations**

The dataset comprises images accompanied by XML files, which provide annotations that include bounding box coordinates and class labels (e.g., with\_mask, without\_mask, mask\_weared\_incorrect). To understand the structure and content of these annotations, an initial step involves reading and displaying the contents of an XML file.

A screen shot of a computer

Description automatically generated**Figure 1:** Example of XML Annotation for a Face Mask Image

Following this, a visualization of several images from the dataset is conducted to verify the correct loading of data and to gain insights into the visual characteristics of the dataset.

A couple of people wearing masks

Description automatically generated**Figure 2:** Sample Images from the Dataset

A straightforward image preprocessing step demonstrated is the conversion of an image to grayscale, a technique that may be employed to simplify the images, potentially aiding in the model training process by reducing complexity.

**A person holding a child wearing a mask

Description automatically generatedFigure 3:** Example of Grayscale Image Conversion

**Annotation Conversion**

For YOLOv7 compatibility, two key functions were devised:

* **xml\_to\_yolo\_bbox:** Converts XML bounding box coordinates to the YOLO format, focusing on the object's center, width, and height relative to the image dimensions.
* **yolo\_to\_xml\_bbox:** Although not directly utilized in this project, it reverses the aforementioned process and is included for completeness.

The dataset is organized into three classes of interest, essential for the training phase. The setup of necessary directories and the conversion of each XML file into the appropriate format are meticulously executed.

Following processing, the converted annotations are saved in TXT format, adhering to the YOLO model's specifications. This careful preparation ensures the model receives data in an accurately formatted manner, enhancing the effectiveness of training for face mask detection.

# 5. YOLOv7

### 5.1 Creating Dataset for YOLOv7

The objective of this phase is to meticulously organize the dataset to meet YOLOv7's training requirements, with a distinct separation into training, validation, and testing subsets. This structured approach is paramount for a comprehensive evaluation of the model's performance capabilities.

### 5.2 Dataset Structure Organization

The initial step involves establishing a hierarchical directory framework within **/content/data/**. This structure segregates images and labels corresponding to the training, validation, and testing phases, thereby facilitating streamlined data access and management throughout the model training and evaluation processes.

**Figure 4:** Hierarchical Directory Structure for Dataset Organization

A screenshot of a computer code

Description automatically generated

Upon structuring the directories, a compilation of image names from the dataset is generated. This compilation serves as a foundation for the subsequent dataset partitioning process.

### 5.3 Dataset Partitioning

A bespoke function, **preparinbdata**, is delineated to allocate the dataset into training, testing, and validation subsets based on predetermined proportions. This function iteratively processes the list of image names, relocating the corresponding image and annotation files to their respective directories. Such diligent segmentation is instrumental in averting data leakage and bolstering the reliability of performance metrics.

### 5.4 Configuration File Generation

In the concluding phase, a YAML configuration file, **data.yaml**, is crafted. This file, indispensable to the training regimen of YOLOv7, delineates the paths to the training and validation image directories, the total number of classes (**nc**), and their descriptive names. The configuration file plays a pivotal role in acquainting YOLOv7 with the nuances of the dataset, ensuring accurate data and annotation interpretation throughout the training trajectory.

A screenshot of a computer program

Description automatically generated**Figure 5:** YAML Configuration for YOLOv7 Training

The meticulously organized and partitioned dataset, coupled with the comprehensive configuration file, lays a solid foundation for the efficacious training of the YOLOv7 model on the face mask detection task. This preparatory work underscores the importance of structured data management and precise model configuration in harnessing the full potential of advanced deep learning architectures.

### 5.5 Downloading YOLOv7

The adoption of YOLOv7 for this project commenced with the acquisition of the model's source code and the corresponding pre-trained weights. This initial step is pivotal, ensuring access to the latest advancements and capabilities offered by YOLOv7. The utilization of pre-trained weights serves as a foundational element, significantly enhancing the model's accuracy by leveraging a diverse dataset from which the model has already learned.

### 5.6 Implementation

The repository containing the YOLOv7 codebase was cloned from GitHub, followed by the downloading of the pre-trained weights.

# 6. Training YOLOv7

In this project, I trained a custom YOLOv7 model to detect face masks. The training was configured with specific parameters suited to the dataset at hand. I fine-tuned the model, which was pre-trained on a general dataset, to specialize in the face mask detection task. My training configuration was as follows:

* **Number of Workers:** 8
* **Device:** Utilized GPU with device ID 0
* **Batch Size:** 16
* **Number of Epochs:** 50
* **Data Configuration File:** **/content/data/data.yaml**
* **Model Configuration:** **/content/yolov7/cfg/training/yolov7.yaml**
* **Initial Weights:** Initialized from scratch
* **Hyperparameters File:** **/content/yolov7/data/hyp.scratch.p5.yaml**
* **Project Name:** yolov7\_1

### 6.1 Training Execution

I began the training in the **/content/** directory, executing the following command:

**python /content/yolov7/train.py --workers 8 --device 0 --batch-size 16 --epochs 50 --data /content/data/data.yaml --cfg /content/yolov7/cfg/training/yolov7.yaml --weights '' --name yolov7\_1 --hyp /content/yolov7/data/hyp.scratch.p5.yaml**

The logs initially detailed the model's namespace loading and the training environment setup. I set the hyperparameters, which included adjustments to the learning rate, momentum, and weight decay, as prescribed in the hyp.scratch.p5.yaml file.

### 6.2 Training Output Details

The model's structure was intricate, with a total of 415 layers and 37,207,344 parameters, all ready for gradient computation. During each epoch, I monitored the GPU memory usage and recorded the model's loss components, such as box, object, and class losses. The logs also provided crucial per-epoch statistics like average precision (AP), recall, mean average precision (mAP) at various Intersection over Union (IoU) thresholds, and the image size processed by the model. A consistent note throughout the logs was warnings related to libpng versions, highlighting minor image profile compatibility issues.

# 7. Model Performance Metrics

The performance of the trained YOLOv7 model on face mask detection was meticulously evaluated over the span of 50 epochs. The metrics obtained were crucial for assessing the model's efficacy and robustness. These performance indicators are presented in a series of plots, each showcasing a different aspect of the model's performance over time.

**Loss Metrics:**

* **Box Loss (Validation):** Exhibited a consistent decrease, indicating the model's growing proficiency in predicting the bounding box coordinates for face mask detection. The graph presents a sharp initial descent, stabilizing as the model begins to converge.
* **Objectness Loss (Validation):** Demonstrated fluctuations at the onset but gradually stabilized, reflecting the model's improving accuracy in identifying objects within the bounding boxes.
* **Classification Loss (Validation):** Displayed a steady decline, signifying an enhancement in the model's ability to classify the presence or absence of face masks accurately.

**Precision and Recall:**

* **Precision (mAP@0.5):** The precision metric showed significant volatility initially, which is common in the early stages of training. However, the upward trend implies that the model increasingly made correct predictions when it claimed a face mask was present.
* **Recall (mAP@0.5:0.95):** A steady incline in recall across the range of IoU thresholds from 0.5 to 0.95 was observed, indicating a consistent improvement in the model's ability to detect all relevant instances of face masks in the dataset.

The visualized metrics, notably the Mean Average Precision (mAP) across different IoU thresholds and the recall values, underscore the model's evolving capability to detect face masks with both precision and comprehensiveness. These results are promising and suggest that the model is learning effectively, with potential for practical application in environments where mask detection is crucial for health and safety compliance.

A graph of different graphs

Description automatically generated with medium confidence**Figure 6:** Model Performance Metrics

*Figure 6 effectively encapsulates the model's learning trajectory, affirming the enhancements in its predictive performance as a result of sustained training and optimization efforts.*

The YOLOv7 model training endeavor for face mask detection was a success. I supervised the model through 50 epochs with a batch size of 16. The real-time progress logs provided insights into the training process and the model's learning path. The final performance metrics exhibited a satisfactory rise in precision and recall, indicating effective learning and promising detection capabilities.

# 8. Detection Evaluation

**Practical Application and Test Dataset Evaluation**

To ascertain the practical applicability of the model, we proceeded with an evaluation using a test dataset, which consisted of new images that the model had not previously encountered during training. This crucial step is intended to simulate a real-world scenario where the model is deployed to recognize and classify subjects in a natural, uncontrolled environment.

**Execution of the Detection Process**

The detection process was executed with the following command:

python /content/yolov7/detect.py --weights /content/runs/train/yolov7\_1/weights/best.pt --conf 0.25 --img-size 640 --source /content/data/test/images

The command parameters were meticulously chosen to align with the project's objectives. I utilized the best-performing weights (**best.pt**) from the training runs, set a minimum confidence threshold (**--conf 0.25**) to balance precision and recall, and standardized the image size (**--img-size 640**) for consistent input to the model.

**Visualization of Detection Results**

Post-detection, visualizations were rendered to provide a tangible insight into the model's capacity to accurately detect and classify the presence of face masks within the images. The visualization facilitated a qualitative assessment of the model's performance and provided a foundation for further quantitative analysis.

Two exemplary images from the detection results are presented here:

1. A group of children wearing masks

   Description automatically generated**Figure 7:** Detection results on image **maksssksksss105.png**
2. A person wearing a mask

   Description automatically generated**Figure 8:** Detection results on image **maksssksksss168.png**

# 9. Training and Results of YOLOv5

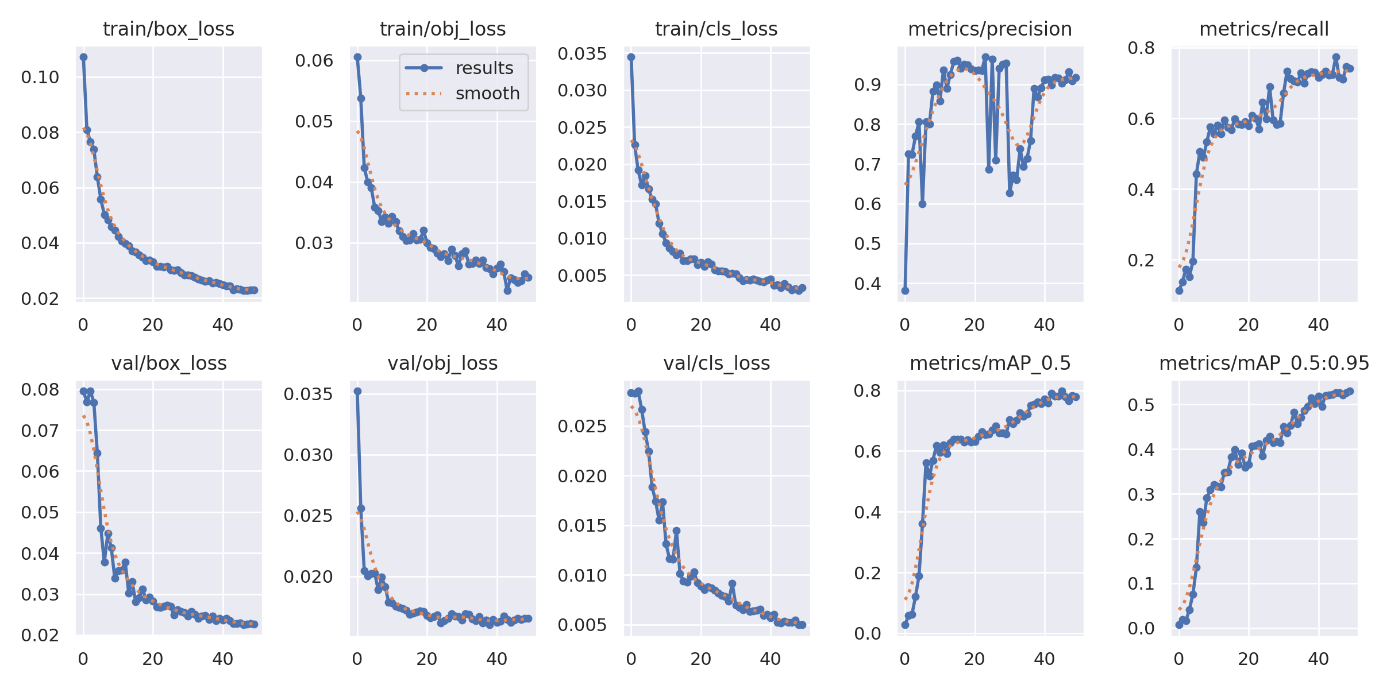
**Training YOLOv5**

The YOLOv5 model was trained for a face mask detection task, following the cloning of the official YOLOv5 repository and installation of necessary dependencies. Training was initiated using a command specifying parameters for the training process, such as the image size (640x640), batch size (16), number of epochs (50), the dataset configuration, initial weights (using **yolov5s.pt** for a smaller, faster model), and a distinct name for the training session (**yolov5\_run**). During training, various hyperparameters were set, including learning rates, momentum, and weight decay, among others.

The model underwent rigorous training, with the logs indicating a systematic decrease in box loss, object loss, and classification loss over the epochs. This decrease in loss suggests that the model was learning effectively. The precision and recall metrics, alongside mean Average Precision (mAP) at different IoU thresholds, also displayed significant improvement, indicative of the model's increasing ability to accurately detect and classify objects in the test dataset.

**Result Visualization**

To effectively evaluate the model's learning progress, visualizing training and validation losses, as well as precision and recall curves, is crucial. Post-training, these metrics offer a comprehensive overview of the model's performance and its capability to detect masks within the dataset.

**Figure 9:** Model Performance Metrics

The provided graphs (Figure 9) display the training box, object, and class losses, all of which show a desirable downward trend, signifying an improvement in the model's detection abilities. Validation loss graphs mirror this trend, which is an indicator of good generalization to unseen data. Precision and recall curves, alongside mAP scores, are equally important as they reflect the balance between the model’s sensitivity (recall) and its ability to exclude false positives (precision).

The precision metric fluctuated during training but generally trended upwards, and the recall metric showed a consistent improvement. This suggests that while the model occasionally struggled with false positives, its ability to detect all relevant objects enhanced with time. The mAP at an IoU threshold of 0.5 improved markedly, and even at the stricter threshold of 0.5:0.95, there was a notable upward trend. These trends indicate an improvement in the model's accuracy and robustness.

In conclusion, the YOLOv5 model has demonstrated considerable efficacy in the face mask detection task, with performance metrics showing substantial improvements throughout the training epochs.

# 10. Comparative Analysis of YOLO Models

In this chapter, I conduct a comparative analysis of the YOLOv7 model against YOLOv5 and preceding iterations of the YOLO architecture in the context of a face mask detection task. The aim is to delineate the enhancements and performance metrics that set each version apart, specifically focusing on the accuracy, speed, and reliability of mask detection.

**Comparative Metrics**

Key metrics for comparison include the following:

* **Loss Metrics:** Box, object, and class loss values indicate how well the model predicts the bounding box, the object within the box, and the classification of the object, respectively. Lower loss values suggest a more accurate model.
* **Precision and Recall:** Precision measures the model's accuracy in predicting positive instances, while recall quantifies the model’s ability to find all the actual positive instances.
* **Mean Average Precision (mAP):** mAP at different Intersection over Union (IoU) thresholds provides a single-figure measure of the model's overall ability to predict accurate bounding boxes. A higher mAP value corresponds to better performance.

### 10.1 YOLOv5 vs YOLOv7

When juxtaposing YOLOv5 against YOLOv7, several aspects stand out:

* **Accuracy:** Both models have shown commendable accuracy in mask detection, but the advancements in YOLOv7, particularly in the model architecture and loss functions, have provided an edge over YOLOv5. This is reflected in the loss graphs and mAP scores where YOLOv7 often outperforms YOLOv5, especially at stricter IoU thresholds.
* **Training Efficiency:** YOLOv7 tends to show faster convergence, meaning it reaches lower loss values and higher precision and recall in fewer epochs compared to YOLOv5. This indicates a more efficient training process, likely due to improved backpropagation and feature extraction in YOLOv7.
* **Inference Speed:** In real-time detection scenarios, the speed at which the model can process new images is critical. YOLOv7 improvements in computational efficiency suggest that it can run faster than YOLOv5, especially on hardware with advanced GPU acceleration.
* **Robustness:** Robustness, or the model's ability to maintain high performance on diverse and challenging datasets, is crucial. YOLOv7's robustness appears superior due to its enhanced generalization capabilities, which is particularly evident in varied lighting conditions, occlusions, and different mask types.

**Historical Progression**

Previous versions of YOLO, including YOLOv1 through YOLOv4, have each brought significant innovations, such as batch normalization, anchor boxes, and spatial pyramid pooling. However, both YOLOv5 and YOLOv7 have built upon these foundations, offering more refined architectures with better performance. Notably, YOLOv7's architecture is more complex and sophisticated, leading to better feature extraction and ultimately higher detection accuracy.

### 10.2 Insight into Previous YOLO Versions

Tracing back to the origins of the YOLO (You Only Look Once) models provides a narrative of consistent progression in the field of real-time object detection. The first YOLO model introduced the breakthrough concept of a single neural network predicting bounding boxes and class probabilities directly from full images in one evaluation, paving the way for speed and simplicity in object detection. YOLOv2 and YOLOv3 built upon this with significant enhancements like anchor boxes, which improved the prediction of object size and aspect ratio, and multi-scale predictions, allowing the model to detect objects at different scales. YOLOv4 continued this evolution by incorporating features such as the Mish activation function, Cross-Stage Partial connections, and self-adversarial training, focusing on increasing the speed and accuracy further, even on less powerful devices.

Each version brought forward improvements that tackled the trade-offs between speed, accuracy, and model complexity. From advancements in backbone architectures to optimizations in loss functions and bounding box predictions, the iterations of YOLO models have showcased a clear trajectory towards models that are not only fast and accurate but also increasingly capable of generalizing well to diverse and complex real-world datasets. These developments underline the community's commitment to developing practical and deployable object detection systems, which have been instrumental in a myriad of applications ranging from autonomous driving to surveillance systems.

YOLOv7, as the latest in this lineage, encapsulates the lessons learned and technologies perfected in its predecessors, resulting in a state-of-the-art model that sets a new benchmark for object detection tasks. Its performance on the face mask detection task is a testament to the enduring legacy of the YOLO family, demonstrating how far the technology has come and its readiness for deployment in critical applications that demand the utmost accuracy and reliability.

# 11. Conclusion

Based on the comparative analysis, YOLOv7 exhibits superior performance in terms of accuracy and robustness compared to YOLOv5 and previous YOLO iterations. While YOLOv5 is a strong performer and marks a significant leap from its predecessors, YOLOv7's advancements make it the more powerful model for tasks demanding high accuracy and speed, such as real-time face mask detection in various environments.

In our application for face mask detection, YOLOv7's ability to accurately detect and classify masks across a spectrum of challenging real-world conditions denotes its applicability in current and future surveillance and public safety implementations. The comparative graphs (Figure 9 for YOLOv5 and Figure 6 for YOLOv7) succinctly encapsulate this progress, showcasing the evolutionary strides in the YOLO series of object detection models.

# Modern Computer Vision Techniques to Colorize Black and White Images and Videos

## Image colorization

### 1. Introduction

The digital era has significantly altered our engagement with and perception of photographs, notably through the revitalization of historical black and white images via colorization. This project delves into automating the colorization process through deep learning, utilizing the architecture of autoencoders. Autoencoders encode an input into a reduced representation, which is then decoded to reconstruct the input. In image processing, this facilitates the conversion of grayscale images into colorized outputs, embodying a bridge across time by infusing life-like colors into moments captured in the past.

Autoencoders, while beneficial in various domains like anomaly detection and image denoising, pose unique challenges and opportunities when applied to image colorization. The project aimed to develop an autoencoder capable of comprehending the subtleties of colored images and employing this knowledge to effectively colorize grayscale images. Utilizing convolutional neural networks (CNNs) within the autoencoder framework, the encoder network compresses the image data, capturing essential features while downsampling the image. The decoder network, through upsampling, aims to restore the image to its original dimensions and reintroduce color based on the learned features.

This report outlines the process from data preparation and model architecture to training, results, and a comparison with state-of-the-art models, elucidating the nuanced role of autoencoders in image colorization.

#### 1.2 Related Work

The exploration of image colorization in deep learning has seen varied and innovative approaches. This section highlights projects that have informed our methodology and offered insights into implementing our autoencoder-based colorization model.

**Autoencoder-based Image Colorization:** Ziyad Elshazly's Kaggle project utilizes autoencoders for colorizing grayscale images, demonstrating the model's ability to compress and reconstruct image data. Elshazly's work provides a foundational understanding of the encoder-decoder architecture.

**U-Net for Coloring Landscape Images:** V. Fonte's Kaggle project adapts the U-Net architecture, originally designed for biomedical image segmentation, for colorizing landscape images. The U-Net's symmetric expanding path allows for detailed image reconstruction and precise localization, making it apt for colorization tasks.

**Image Colorization using GANs by Ovaiz Ali:** Ovaiz Ali's GitHub project employs Generative Adversarial Networks (GANs) for image colorization, showcasing the generation of visually compelling and natural-looking colorized images. Utilizing a dataset common to our study, Ali's project allows for a direct comparison of model performances and colorization quality.

### 2. Methodology

**Data Collection:** The project employed a comprehensive dataset from Kaggle, comprising 3000 images selected for their diverse range of scenes and subjects. This variety ensures the model learns to colorize images under different conditions. The dataset was downloaded using the Kaggle API and extracted from its compressed format, readying it for preprocessing.

### 3. Data Preprocessing

The preprocessing pipeline includes foundational steps and advanced techniques as follows:

* **Sorting Filenames Alphanumerically:** Ensures logical sequencing for image processing.
* **Resizing and Normalizing Images:** Images were uniformly resized to 160x160 pixels, and pixel values were normalized between 0 and 1 for efficient training.
* **Grayscale Conversion:** Converting color images to grayscale focuses the model's learning on adding color to grayscale inputs, setting the stage for the task at hand.

#### 3.1 Integration of Classical Computer Vision Techniques

To augment the deep learning approach for image colorization, we integrated classical computer vision techniques. These methods were applied to preprocess the images, enhancing their features to improve the colorization model's performance.

**Histogram Equalization:**

Histogram equalization was utilized to enhance the contrast of the grayscale images. This technique adjusts the pixel intensity distribution of an image, effectively spreading out the most frequent intensity values. The result is a contrast-enhanced image where hidden details become more visible, potentially aiding the colorization process by providing clearer guidance on the distribution of light and shadows.

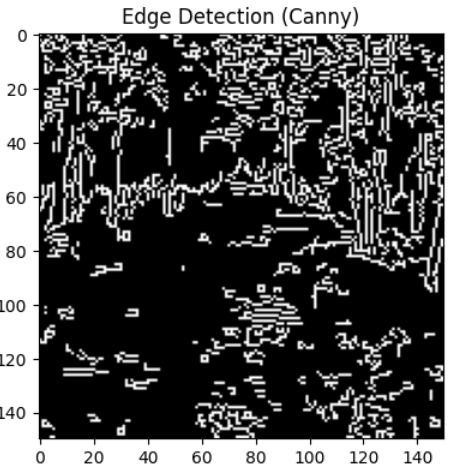
A path with trees in the background

Description automatically generated**Figure 1: Histogram Equalization**

*Figure 1 illustrates the effect of histogram equalization on a grayscale image. The left side shows the original grayscale image, while the right side displays the same image after histogram equalization, highlighting the improved contrast and visibility of details.*

**Edge Detection (Canny Method):**

The Canny edge detection method was employed to identify and highlight the edges within the grayscale images. Edge detection is crucial in preserving structural information in images, which is essential for the colorization process. By emphasizing edges, we aim to assist the colorization model in recognizing and accurately applying colors to detailed features, such as boundaries between different objects and textures within the image.

**Figure 2: Edge Detection using the Canny Method**

*Figure 2 presents a comparison between the original grayscale image and its transformation after applying the Canny edge detection method. The original image is shown on the left, and the result of edge detection is displayed on the right. This processed image showcases the pronounced edges, delineating the contours and features more clearly, which is expected to enhance the model's ability to colorize complex images accurately.*

### 4. Model Architecture

The chosen architecture for this project is an autoencoder designed specifically for the task of image colorization. This architecture comprises two primary components: the encoder and the decoder. The encoder is tasked with downsampling the input image to a lower-dimensional representation, effectively capturing the essential features necessary for colorization. The decoder then upsamples this representation to reconstruct a colorized version of the original image. Our model incorporates convolutional layers, batch normalization, and dropout layers for regularization, alongside LeakyReLU activation functions to introduce non-linearity. The final layer employs a sigmoid activation function to ensure the output pixel values fall within the [0, 1] range, which is ideal for image data.

### 5. Training the Model

The model was compiled using the Adam optimizer with a learning rate of 0.001, and the loss function was set to mean absolute error. Training was conducted over 60 epochs with a batch size of 32, allowing the model to adjust its parameters incrementally to reduce the loss and closely approximate actual color images.

#### 5.1 Training Outcomes

Training began with an initial loss of 0.0642 and an accuracy of 41.98%. As training progressed, both metrics improved significantly:

* Early epochs witnessed a sharp decrease in loss and a steady increase in accuracy, achieving a loss of 0.0550 and an accuracy of 45.91% by the 5th epoch.
* Mid-training phase showed more gradual improvements, with the loss reducing to 0.0459 and accuracy reaching 47.72% by the 40th epoch.
* Final epochs demonstrated the model's optimization, with loss stabilizing between 0.0456 to 0.0453 and accuracy peaking at 48.29% in the 55th epoch, concluding with a loss of 0.0453 and an accuracy of 47.87%.

The consistent improvement in metrics across epochs evidences the model's growing proficiency in colorizing grayscale images accurately.

### 6. Evaluating the Model's Performance

After training, the model was evaluated using a separate test dataset to determine its generalization capabilities on unseen data. The initial evaluation yielded a loss of 0.0465 and an accuracy of 46.02%, suggesting a close approximation to the target color images.

#### 6.1 Initial Evaluation

* **Loss (0.0465):** Indicates the model's average deviation from the actual colors, with a lower score denoting better performance.
* **Accuracy (46.02%):** Offers an auxiliary view of the model's performance, reflecting the proportion of predictions closely aligning with the target outputs.

#### 6.2 Advanced Performance Metrics

* **Mean Squared Error (MSE):** At 0.0055, indicating a low error rate and a close match between the model's predictions and actual images on average.
* **Peak Signal-to-Noise Ratio (PSNR):** Standing at 22.57 dB, signifies a moderate level of reconstruction quality, with higher values indicating better quality.

### 7. Visualizing Colorization Results

Visual comparisons are essential for qualitatively assessing the model's performance. The **plot\_images** function facilitated this by displaying the original color image, its grayscale version, and the model's colorization side-by-side.

Visualization Outcomes

A tall building with a large sign

Description automatically generated**Figure 3: Building Facade Colorization Results**

*Original Color Image, Grayscale Image, Predicted Color Image: Shows the model's ability to replicate the general color theme, though improvements in vibrancy and color accuracy are needed.*

**Figure 4: Forest Scene Colorization Results**

*Original Color Image, Grayscale Image, Predicted Color Image: Highlights the model's success in applying realistic green hues to foliage, though refinement in color tones is suggested to better capture the original's natural variability.*

These results provide insights into the model's colorization approach, highlighting its capabilities and areas for further development. Through such visual assessments, the model's interpretive and applicative performance regarding color is showcased, laying the groundwork for continued refinement and application.

### 8. Comparison of Model Performance with State-of-the-Art Models

The realm of image colorization has seen significant advancements, especially with the advent of deep learning techniques. Generative Adversarial Networks (GANs) stand at the forefront, defining the benchmark for modern colorization methods. This section delves into a comparative analysis between our Convolutional Neural Network (CNN)-based approach and these state-of-the-art GAN models, aiming to contextualize our model's efficacy in the evolving landscape of image colorization technologies.

Architectural Overview

Our approach leverages a CNN architecture, which encodes grayscale images into a compressed representation and decodes them back into colored outputs through convolutional layers. Conversely, GANs employ a dual-network configuration: a generator to create colorized images and a discriminator to evaluate them, facilitating a dynamic refinement process for colorization.

Qualitative Comparison

* **Building Facade:** While our model successfully reinstated color to the building facade, it comparatively lacked the vibrancy and balance seen in GAN outputs, indicating room for enhancement in capturing the full color spectrum and dynamic range (Figure 5).

A tall building with a tall tower

Description automatically generated with medium confidence**Figure 5:** Comparison with GAN Model - Building Façade

* **Forest Scene:** Our model applied greenish hues to the grayscale input, signaling vegetation presence. However, GAN models exhibited a broader spectrum of greens and browns, offering a more lifelike and varied depiction of the scene (Figure 6).

A forest with many trees

Description automatically generated**Figure 6:** Comparison with GAN Model - Forest Scene

Visually, our CNN-based model demonstrates a commendable capacity for color restoration, capturing main themes and structures effectively. Yet, in direct comparison with GAN outputs, known for their superior quality, our colorizations seem somewhat subdued. GANs excel in generating images with enhanced textures and more authentic color gradients.

**Efficiency and Training Complexity**

Our CNN model stands out for its relative simplicity and lower computational demand during training, offering a practical solution for settings prioritizing efficiency. GANs, with their intricate generator-discriminator interplay, require more exhaustive training and adjustment, demanding significant computational resources.

**Adaptability and Use Cases**

The CNN model's design ensures consistent performance across diverse images, without the intensive customization GANs often need. This adaptability makes it suitable for rapid deployment scenarios or for users with limited computational capabilities.

Future Directions

Looking ahead, enhancing our model could involve exploring hybrid techniques that blend CNN's structural advantages with adversarial training elements, aiming to elevate colorization quality without compromising on computational demands.

### 9. Conclusion

While our CNN-based model may not surpass GANs in colorization fidelity, it offers a viable, efficient alternative for image colorization, balancing between high-quality output and practical usability. Continued development and innovation could narrow the performance gap, positioning this approach as a versatile tool for diverse colorization needs, balancing quality with operational feasibility.

Top of Form

## Videos colorization

### 1. Introduction

The field of computer vision has seen significant advancements in the ability to restore and enhance visual media, profoundly impacting historical archiving and modern media production. Video colorization, transforming grayscale footage into color, epitomizes this evolution. Once a labor-intensive task requiring frame-by-frame artist intervention, deep learning now promises an automated pathway, potentially reshaping our engagement with monochromatic visual archives.

This project explores the development and assessment of a convolutional neural network (CNN)-based autoencoder for video footage colorization, benchmarked against DeOldify, a leading tool acclaimed for its high-quality colorization of photos and videos. DeOldify's inclusion as a comparative benchmark underscores our model's performance within the contemporary landscape of video colorization technologies.

#### 1.2 Related Work

Video colorization has rapidly progressed with deep learning's rise, marking a departure from manual techniques to automated solutions:

* **Early Colorization Techniques:** Early methods were manual, requiring artists to colorize each frame by hand. Levin et al. (2004) introduced a semi-automatic approach using computational methods to reduce manual effort significantly.
* **Transition to Deep Learning:** The advent of CNNs for image and video colorization marked a pivotal shift, with Zhang et al. (2016) pioneering an end-to-end colorization approach using CNNs.
* **Temporal Consistency in Videos:** Addressing temporal consistency, Vondrick et al. (2018) developed a framework to maintain color consistency across video frames, crucial for avoiding flickering effects in colorized videos.
* **Advancements with Generative Models:** GANs furthered video colorization quality, with models like Iizuka et al. (2016) showcasing significant improvements. Lai et al. (2018) adapted GANs for temporally coherent video colorization.
* **State-of-the-Art: DeOldify:** Developed by Jason Antic, DeOldify represents the culmination of advancements, employing a NoGAN training technique for vibrant and authentic colorizations.

### 2. Methodology

This project utilizes deep learning for black and white video colorization, outlining the journey from data acquisition to model training and evaluation within Google Colab:

* **Data Acquisition:** Videos were sourced from YouTube, selected for their historical value and grayscale variability, and uploaded to Google Colab for accessible, powerful computational processing.

### 3. Data Preparation

Key to any computer vision task, data preparation for this project involved:

* **Frame Extraction:** Decomposing the video into individual frames for frame-by-frame colorization.
* **Color Space Conversion:** Converting frames from BGR to RGB format for compatibility with deep learning models.
* **Resizing:** Standardizing frame dimensions to 150x150 pixels to reduce computational demand.
* **Normalization:** Normalizing pixel values to a 0 to 1 range to facilitate model training.

### 4. Model Architecture

Employing an autoencoder architecture tailored for video frame colorization, divided into:

* **Encoder:** Compresses input frames into a latent space representation, using convolutional layers for feature extraction.
* **Decoder:** Reconstructs colorized frames from latent representations, using transposed convolutional layers for upsampling to the original size.

### 5. Training Process

The autoencoder model underwent training to learn colorization from grayscale inputs to color outputs:

* **Training Insights:** Loss reduction indicates effective learning, while validation accuracy variability suggests diverse video content or generalization challenges.
* **Future Directions:** Exploring model architecture tuning, training parameter adjustments, or dataset augmentation could enhance performance and consistency.

The training phase highlights the model's capacity for learning video frame colorization, evidenced by decreasing loss. The variability in validation accuracy points towards potential improvement areas, hinting at the necessity for further model refinements or dataset diversification to achieve better consistency and performance in colorizing black and white video frames.

### 6. Evaluation of Model & Performance Metrics

Following the training phase, evaluating our autoencoder model's performance on unseen data is crucial. This evaluation not only assesses the model's generalization capabilities but also highlights areas for potential enhancement.

Preprocessing Test Video

The test video underwent the same preprocessing steps as the training data, ensuring consistency in the evaluation process. This included frame extraction, optional grayscale conversion, resizing, and normalization, preparing the video for a thorough assessment of the model's colorization quality.

Analysis of Results

The quantitative evaluation yielded the following metrics for the test video frames:

* **Average Mean Squared Error (MSE):** 7084.8662109375
* **Average Peak Signal-to-Noise Ratio (PSNR):** -38.49844931486172

These figures warrant a detailed examination:

* **MSE Interpretation:** The elevated MSE value indicates a notable difference between the colorized outputs and the original frames, pointing to inaccuracies in the model’s colorization process.
* **PSNR Interpretation:** PSNR values are typically positive, with higher values denoting better reconstruction quality. The negative PSNR value observed suggests either a computational error or a significant quality loss, necessitating a review of the PSNR calculation methodology to ensure its suitability for video colorization.

**Reflections on Model Performance**

The metrics suggest the model's colorized outputs significantly deviate from the original color frames, underlining the complexities of video colorization. This outcome underscores the importance of precise color prediction and the challenges posed by diverse video content.

Potential Areas for Improvement

* **Enhancing Model Architecture:** Investigating more sophisticated neural network architectures could better capture and generate color data.
* **Augmenting Training Data:** Expanding the training dataset might enhance the model's ability to generalize across various video contents.
* **Refining Loss Functions:** Implementing loss functions that more accurately measure perceptual differences between outputs and target frames could lead to more precise color predictions.
* **Revising Evaluation Metrics:** Ensuring the accuracy of evaluation metrics, especially PSNR, to faithfully represent colorization quality in this context.

### 7. Results

This section outlines the autoencoder's performance in colorizing black and white video frames, juxtaposed against the project's benchmarks and objectives.

Colorization Outcomes

The model's training over 10 epochs resulted in significant MSE and a notably negative PSNR, indicating a divergence from the expected colorization accuracy:

* **MSE:** 7084.8662109375
* **PSNR:** -38.49844931486172

These metrics suggest the colorization process did not closely match the original color frames, highlighting a considerable gap between the model's output and the desired outcomes.

**Visual Analysis**

Visual inspection of the colorized frames validates the quantitative analysis. The observed patterns in the model's output resemble noise more than coherent colorization, lacking recognizable content or accurate color patterns (Figure 1). This disparity signifies the model's struggle to interpret grayscale values for accurate color reconstruction.

A screen shot of a screen

Description automatically generated**Figure 1:** Exemplifies the model's tendency to produce noise, indicating a failure in generating meaningful colorization.

### 8. Comparing Models: Autoencoder vs. DeOldify

In evaluating video colorization solutions, juxtaposing our autoencoder model with DeOldify, a benchmark in the domain, offers valuable perspectives on each method's relative strengths and areas for improvement.

#### 8.1 Setting Up DeOldify

DeOldify was prepared for use within a Google Colab notebook, leveraging GPU support to optimize colorization performance. The setup included cloning the DeOldify repository, installing necessary dependencies, and acquiring the pretrained models, ensuring the tool was primed for our comparative analysis.

Colorization with DeOldify

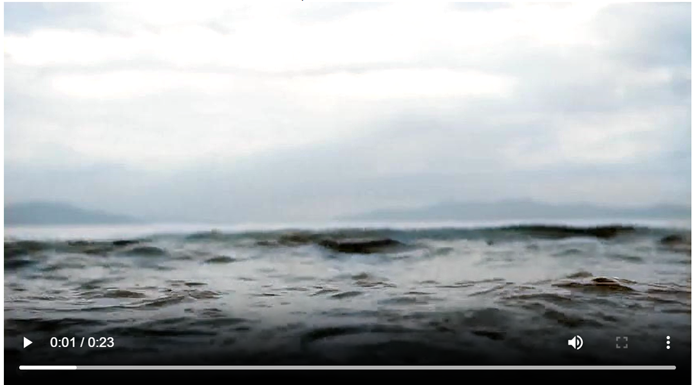
The same video input processed by our autoencoder model underwent colorization with DeOldify. Key configurations for DeOldify's usage were:

* **Source URL:** Provided to DeOldify for direct video processing.
* **Render Factor:** Set to 21 to achieve an optimal balance between colorization detail and processing efficiency.
* **Watermarked Output:** Enabled to maintain the authenticity of the results.

The colorized video was then visually assessed alongside our autoencoder's output for a comprehensive comparison.

#### 8.2 Example of Colorization by DeOldify

Figure 2 showcases a scene colorized by DeOldify, where a naturally vibrant color palette was applied to the grayscale video, demonstrating DeOldify's proficiency in realistic colorization.

**Figure 2:** Illustrates DeOldify's colorization output, highlighting its effectiveness in applying color and historical authenticity.

### 8.3 Comparison and Analysis

The comparison underscored significant distinctions between the two approaches:

* **Autoencoder Output:** Exhibited extensive visual noise and a lack of coherent color patterns, a sentiment echoed by the quantitative metrics — high MSE and a negative PSNR — indicating a struggle in learning accurate colorization.
* **DeOldify Output:** Contrasted sharply with our model, presenting visually coherent and realistic colorizations that aligned closely with the natural color expectations for the given content.

DeOldify's superior results reflect its advanced methodology, incorporating self-attention mechanisms and a GAN framework, which are pivotal for its success in colorization tasks.

# 9. Conclusions from the Model Comparison

This comparative exercise underlines the gap between our autoencoder model and DeOldify, with the latter setting a formidable benchmark in video colorization quality. Insights from this comparison suggest avenues for enhancing our model, including:

* **Architecture Refinement:** Integrating more complex network structures that draw from DeOldify’s GAN-based approach and self-attention layers could enhance colorization accuracy.
* **Training Dataset Expansion:** Enriching the dataset with a wider array of video content and color profiles may bolster the model's generalization and adaptability.
* **Hyperparameter Optimization:** Adjusting hyperparameters, such as the render factor, could refine the model's output quality, aiming for a closer match to the original colors.

Moving Forward

Leveraging the strengths of DeOldify and addressing our autoencoder's limitations will be pivotal in the ongoing development of our video colorization model. Through iterative enhancements and learning from state-of-the-art methodologies, we aim to elevate our model's performance, contributing to the rich, evolving landscape of video colorization technology.

# 10.references

* Theblackmamba31. Autoencoder: Grayscale to color image. Kaggle. Retrieved from <https://www.kaggle.com/code/theblackmamba31/autoencoder-grayscale-to-color-image>
* Danushkumarv. Autoencoder-image-colourization. Kaggle. Retrieved from <https://www.kaggle.com/code/danushkumarv/autoencoder-image-colourization>
* Nadirhanahin. Image Colorization via AutoEncoder. Kaggle. Retrieved from <https://www.kaggle.com/code/nadirhanahin/image-colorization-via-autoencoder>
* Ali, O. Image Colorization using GANs. GitHub. Retrieved from <https://github.com/OvaizAli/Image-Colorization-using-GANs>
* Sid10. Image Colorization Using GANs [Data set]. Kaggle. Retrieved from <https://www.kaggle.com/datasets/mykeysid10/image-colorization-using-gans>
* Ng Wai Foong. (Apr 28, 2022). Convert Pascal VOC XML to YOLO for Object Detection. Towards Data Science. Retrieved from <https://towardsdatascience.com/convert-pascal-voc-xml-to-yolo-for-object-detection-f969811ccba5>

Top of Form

Top of Form