PBL Research Report

Zeyad Mohamed Ibrahim 120210112
Ahmed Mohamed Ahmed Hassan 120210113
Ahmed Yasser Gamea 120210114
Zeyad Aly Reslan 120210115
Yousef Wael Gaber 120210121
Abdelrahman Amr Elsayed 120210128
Abdelaziz Essam Abdallah 120210130
Omar Ahmed Mostafa 120210131
Mohamed Aref Attia 120210132
Hesham Abdalgawad 120210135

June 5, 2024

Contents

1 Abstract **4**

2 GANs papers **6**
2.1 Generative Adversarial Networks: Introduction and Outlook . . . 6
2.2 Traditional Methods for Image SR . . . 8
2.3 A Survey on Image Super-Resolution with Generative Adversarial Networks . . . 9
2.4 Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (SRGANs) . . . 11
2.5 Best-Buddy GANs for Highly Detailed Image Super-Resolution (BEBY GANs) . . . 12
2.6 PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION . . . 13
2.7 Supervised Pixel-Wise GAN for Face Super-Resolution . . . 15
2.8 Improving GAN Performance in Single Medical Image Super-Resolution . . . 16
2.9 To learn image super-resolution, use a GAN to learn how to do image degradation first . . . 17
2.10 Multi-Perspective Discriminators-Based Generative Adversarial Network for Image Super Resolution . . . 18
2.11 ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks . . . 19
2.12 Blind Super-Resolution Kernel Estimation using an Internal-GAN . . . 21
2.13 Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data . . . 22
2.14 R-SRGAN: Residual Super-Resolution Generative Adversarial Networks . . . 23

3 Computer Vision Papers **25**
3.1 Simple Online and Realtime Tracking with a Deep Association Metric . . . 25
3.2 StrongSORT: Enhancing Multi-Object Tracking with AFLink and GSI . . . 27
3.3 Towards Real-Time Multi-Object Tracking . . . 28
3.4 YOLO (You Only Look Once) . . . 29
3.5 Machine-learning-based Top-View Safety Monitoring of Ground Workforce on Complex Industrial Sites . . . 30
3.6 YOLO-based Human Action Recognition and Localization . . . 32
3.7 YOLO "YOLO9000" (You Only Look Once) . . . 34
3.8 A Review of YOLO Algorithm Developments . . . 35
3.9 A Novel Real-Time Multiple Objects Detection and Tracking Framework for Different Challenges . . . 36
3.10 You Only Look Once: Unified, Real-Time Object Detection . . . 38
3.11 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision . . . 39
3.12 Introduction to ChainerCV: A Computer Vision Software Library . . . 40
3.13 Literature Review: Real-time Object Detection and Tracking Techniques . . . 42
3.14 CVNets: An Open-Source Library for High-Performance Deep Neural Network Training . . . 44

4 Comparison Between SR Models **46**
4.1 PGGAN . . . 46
4.2 ESRGAN: Enhanced Super-Resolution Generative Adversarial Network . . . 47
4.3 Degradation GAN . . . 48

5 Comparison between Computer Vision Models **50**
5.1 MobileNet SSD . . . 50
5.1.1 Architecture . . . 50
5.1.2 Speed and Efficiency . . . 51
5.1.3 Object Detection . . . 51
5.1.4 Model Size . . . 51
5.1.5 Why Choose MobileNet SSD . . . 52
5.2 YOLOV8 . . . 52
5.2.1 Architecture . . . 52
5.2.2 Speed and Efficiency . . . 53
5.2.3 Object Detection . . . 53
5.2.4 Model Size . . . 53
5.2.5 Why Choose YOLOV8 . . . 54

6 Methodology **56**
6.1 Seminar Work (Research) . . . 56
6.1.1 Super-Resolution Methodology . . . 56
6.1.2 Computer Vision Methodology . . . 58
6.2 PBL Work (Implementation) . . . 60
6.2.1 Overview . . . 60
6.2.2 FireBase . . . 61
6.2.3 Streaming . . . 66
6.2.4 YOLO and AI . . . 69
6.2.5 Vision Transformers . . . 76
6.2.6 Mobile App . . . 87

7 Results and Discussion (Seminar Part - Research) **93**
7.1 Super-Resolution . . . 93
7.1.1 Simple GAN . . . 93
7.1.2 Results . . . 94
7.1.3 WGAN . . . 94
7.1.4 ESRGAN . . . 95
7.2 Computer Vision . . . 97
7.2.1 Confusion Matrix . . . 97
7.2.2 Confusion Matrix (True Positives) . . . 97
7.2.3 Gloves Detection (True Positives) . . . 97
7.2.4 Goggles Detection (True Positives) . . . 98
7.2.5 Helmets Detection (True Positives) . . . 98
7.2.6 Vests Detection (True Positives) . . . 98
7.2.7 Training Progress and Loss Convergence . . . 98
7.2.8 Graph Analysis . . . 99
7.2.9 Decreasing Loss and Model Learning . . . 99
7.2.10 Convergence to 0.02 . . . 99
7.2.11 Implications for Model Accuracy . . . 99
7.2.12 Model Generalization . . . 99
7.2.13 Metrics Analysis: Precision and Recall at Epoch 50 . . . 99

8 Results and Discussion (PBL Part - Implementation) **102**
8.1 YOLO and AI . . . 102
8.1.1 Model Testing and Performance . . . 102
8.1.2 Strengths and Limitations . . . 103
8.1.3 Future Work . . . 103
8.2 Object Detection and Firestore Integration . . . 103
8.2.1 Detection Results . . . 104
8.2.2 Firestore Update . . . 104
8.3 Vision Transformers . . . 105
8.3.1 Evaluation Results . . . 105
8.3.2 Explanation . . . 105
8.4 Mobile App . . . 106

8.4.1 Home Page 106
8.4.2 Notifications Page 107
8.4.3 Real-Time Camera View Page 107
8.4.4 Firebase Database Integration 108

Chapter 1

Abstract

This project presents an integrated system merging artificial intelligence (AI) with a mobile application, facilitated by Google's Firebase for streamlined data management. The primary focus is on enhancing industrial safety through real-time object detection and monitoring. The AI component employs the YOLOv8 model, renowned for its efficiency in annotating objects, while the mobile application serves as an intuitive interface for safety surveillance.

One crucial aspect of the AI system is the implementation of algorithms for checking leg positions and the presence of specific body parts. These algorithms play a pivotal role in ensuring safety compliance by monitoring individuals' positions and protective gear usage in real time.

Additionally, the project delves into the exploration of Vision Transformers, specifically DETR, for object detection accuracy. This exploration provides insights into advanced techniques that can further enhance the system's capabilities.

The results and discussions highlight the superiority of YOLOv8 in real-world scenarios, demonstrating impressive performance in detecting safety equipment like helmets, gloves, vests, and goggles across diverse conditions. The integration of Mediapipe for pose estimation enhances accuracy in safety compliance monitoring, while seamless integration with Firebase Firestore ensures efficient data storage and real-time monitoring.

Strengths of the system include high accuracy, robust performance, and efficient real-time processing, making it suitable for industrial safety applications. However, challenges such as low-light conditions and fixed threshold values are acknowledged as limitations that require further refinement.

Future work will focus on addressing these limitations and exploring advanced techniques for continuous improvement. The integration with Firebase guarantees real-time updates of detection results, including object counts and timestamps, reinforcing the system's reliability and effectiveness in ensuring safety compliance.

In conclusion, this project presents a comprehensive and robust system that integrates AI, mobile

applications, and Firebase for industrial safety monitoring, with a clear roadmap for future enhancements and advancements.

Chapter 2

GANs papers

2.1 Generative Adversarial Networks: Introduction and Outlook

Abstract

This paper introduces Generative Adversarial Networks (GANs) as a potent class of generative models, emphasizing their origin as a two-player zero-sum game for simultaneous learning of a generator and a discriminator. GANs have significantly impacted various AI fields, particularly in image and vision applications, speech processing, malware detection, and chess game programs. The paper surveys the theoretical foundations, implementation models, and applications of GANs, discussing their advantages, disadvantages, and development trends. The proposal background highlights the surge in AI development, the accumulation of generative models, and the deepening of neural networks. The paper concludes by outlining the research levels of AI and the potential of GANs in advancing AI understanding, generation, and creation.

Introduction

The paper discusses the deepening success of neural networks in the past decade, facilitated by increased computing power and data scale. It highlights the flexible structure of neural networks, their applicability in various domains, and the success of the adversarial idea in machine learning. Theoretical foundations, learning methods, and variants of Generative Adversarial Networks (GANs) are introduced, emphasizing improvements and applications. The paper also addresses challenges like the vanishing gradient problem and proposes solutions such as Wasserstein GAN and Loss-Sensitive GAN. Variants like Semi-GAN and Bidirectional GANs are presented, showcasing advancements in GAN modeling. InfoGAN is introduced for capturing mutual information between latent variables and observations.

Applications of GANs

Generative Adversarial Networks (GANs) exhibit versatile applications in various domains. In image and vision computing, GANs generate realistic images, as demonstrated by SRGAN for super-resolution and BEGAN for diverse face samples. GANs contribute to autonomous driving by generating realistic driving scenarios, and SimGAN aids eye detection by bridging synthetic and real image datasets. Image-to-image translation, exemplified by CycleGAN, and frontal view synthesis, such as TP-GAN and StackGAN, showcase GANs' adaptability. In speech and language processing, GANs create dialogue relevance, realistic sentences, and contribute to speech enhancement (e.g., SEGAN). GANs connect with reinforcement learning, imitation learning, and actor-critic methods. Notably, GANs advance malware detection evasion (MalGAN), style transfer (STGANs), and synthetic electronic health records (medGAN). These applications underscore GANs' broad impact, addressing challenges in diverse fields.

Significance and Advancements

Generative Adversarial Networks (GANs) have profound significance in generative models, allowing the creation of diverse, interpretable, and high-dimensional data. Their neural network structure accommodates various loss functions, enhancing model design flexibility. GANs eliminate the need for inefficient Markov chain methods, offering efficient training through backpropagation. They improve sample generation efficiency, diversity, and human interpretability. GANs also prove instructive for semi-supervised learning, using unlabeled data for pre-training. Despite successes, challenges like mode collapse persist. Wasserstein GAN and other variants address stability issues, while ongoing research explores complete mode avoidance. GANs' broader application in parallel intelligence aligns with theories of parallel vision, control, and learning. Future directions involve overcoming limitations and integrating GANs with emerging AI methods.

2.2 Traditional Methods for Image SR

Traditional image super-resolution methods often rely on interpolation techniques to enhance the resolution of an image. Interpolation is a mathematical process that estimates values between known data points. In the context of image super-resolution, interpolation is employed to estimate pixel values for the high-resolution image based on the information available in the low-resolution version.

The process typically involves the following steps:

1. **Pixel Selection:** Given a low-resolution image, the method identifies a pixel or a set of pixels for which high-resolution values need to be estimated.

2. **Interpolation Basis Function:** An interpolation basis function is chosen to determine how neighboring pixel values contribute to the estimation of the target pixel. Common interpolation functions include bilinear interpolation and bicubic interpolation.

3. **Calculation of Gray Values:** The chosen interpolation basis function is applied to the neighboring pixels in the low-resolution image, and their gray values are calculated. These gray values are then combined to estimate the gray value of the target pixel in the high-resolution image.

4. **Reconstruction:** This process is repeated for each pixel in the low-resolution image, resulting in a set of estimated values for the corresponding high-resolution image.

While traditional interpolation methods provide a straightforward way to increase the pixel density in an image, they have limitations. They may not capture complex patterns and details present in the high-resolution version, and the enhancement is often limited by the quality of the available low-resolution information. This is why more advanced techniques, including those based on deep learning, have gained popularity for image super-resolution as they can learn intricate mappings between low and high-resolution images from a large dataset, leading to more effective and realistic results **ref01**.

2.3 A Survey on Image Super-Resolution with Generative Adversarial Networks

This paper delves into the realm of super-resolution methods, with a specific focus on the effectiveness of Generative Adversarial Networks (GANs). Super-resolution is a technique aimed at enhancing low-resolution images with minimal loss, employing processes like denoising and deblurring. The study addresses the growing popularity of GANs, particularly their dual-network architecture, competitive dynamics, and pivotal role in artificial data creation and data enhancement.

The historical evolution of super-resolution techniques is traced, highlighting the transition from Fourier-transform-based methods to neural network solutions. Initially applied to fingerprint image resolution, these neural networks, exemplified by SRGAN (Ledig et al., 2016) and ESRGAN (Wang et al., 2018), have surpassed traditional interpolation methods in restoring high-frequency features. GAN-based studies consistently outperform conventional approaches, showcasing their efficacy in handling super-resolution challenges.

SRGAN, the pioneer in using GANs for realistic super-resolution, places emphasis on perceptual and content loss functions for quality enhancement. SRPGAN excels in small object recognition through semantic analysis. FCGAN, designed for face augmentation, stands out for its preprocessing-free operation and resilience to accessories. ESRGAN builds upon SRGAN by introducing adversarial and perceptual loss functions, with the novel RRDB enhancing natural visual patterns. ESRGAN's superiority is evidenced by its outperformance in the PIRM2018-SR Challenge compared to SRGAN.

Performance metrics are crucial for evaluating super-resolution images. Mean Opinion Score (MOS) provides a subjective assessment of image similarity based on user opinions. Peak Signal-to-Noise Ratio (PSNR) measures noise distortion to quantify image similarity, particularly useful for analog systems. Structural Similarity Index (SSIM) assesses structural resemblance, considering brightness, contrast, and structural attributes. Dark Channel Ratio (DCR), though less common in super-resolution, measures sharpness and clarity.

Perceptual loss functions, as seen in SRGAN, focus on higher-level features during training, improving image quality by addressing limitations in pixel-space assessments. However, they may extend processing times. Pixel-wise loss functions, like Mean Squared Error, calculate errors in pixel space efficiently but may lack visual appeal. Charbonnier Loss enhances low- frequency features, ensuring image accuracy. Feature Matching Loss and Texture Matching Loss prioritize feature-based and style consistency, contributing to realistic results. The choice of loss metric depends on the specific goals and trade-offs between computational efficiency and visual quality in super-resolution image generation.

In conclusion, this study fills a gap by investigating GAN-based super-resolution, recognizing GANs' significance in generating realistic data and enhancing images. GAN applications extend beyond super-

resolution, offering noise reduction in diverse datasets. MOS is proposed as a more accurate perceptual performance metric, considering the limitations of traditional metrics like PSNR and SSIM in capturing visual quality.

2.4 Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (SRGANs)

The goal of single image super-resolution (SR) is to combine two low-resolution images to create a high-resolution image. While deep convolutional neural networks (CNNs) have made significant strides in SR, achieving photo-realism at large upscaling factors remains challenging. This research proposes a unique framework, SRGAN, to achieve photo-realistic single picture super-resolution using a generative adversarial network (GAN).

SRGAN consists of two main components: the discriminator network and the generator network. The generator, based on a deep residual network (ResNet) with skip connections, learns a mapping from the low-resolution space to the high-resolution space to super-resolve the input. The discriminator, also a ResNet, acts as an antagonistic critic, discerning between super-resolved images produced by the generator and actual high-resolution images.

The generator in SRGAN is driven by a comprehensive loss function designed to enhance the quality and fidelity of the reconstructed high-resolution (HR) images from low-resolution (LR) inputs. This multifaceted loss function includes a pixel-wise loss, utilizing Mean Squared Error (MSE), to measure the accuracy of the generated HR image in comparison to the ground-truth HR image. Simultaneously, the adversarial loss incentivizes the generator to craft realistic images that can deceive the discriminator. The perceptual loss leverages features extracted from pre-trained Convolutional Neural Networks (CNNs) like VGG16 or VGG19, ensuring that the super-resolved image retains high-level perceptual qualities.

Conversely, the discriminator's objective is to effectively differentiate between genuine HR images and those synthesized by the generator. The discriminator loss function comprises Binary Cross-Entropy Loss, evaluating the discriminator's accuracy in classifying real and fake images. This loss penalizes the discriminator for misclassifications, compelling it to refine its discrimination capabilities.

The interplay between these intricate loss components for both the generator and discriminator is fundamental to the success of SRGAN in achieving superior single-image super-resolution results. SRGAN demonstrates state-of-the-art performance on benchmark datasets, achieving significant improvements in terms of both quantitative metrics (e.g., Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM)) and subjective visual quality compared to existing SR methods. Notably, SRGAN achieves photo-realistic results even for large upscaling factors (e.g., 4x).

Despite its success, SRGAN introduces computational complexities and substantial memory requirements due to adversarial training, posing implementation challenges.

2.5 Best-Buddy GANs for Highly Detailed Image Super-Resolution (BEBY GANs)

Reconstructing a high-resolution (HR) image from its low-resolution (LR) counterpart is the primary goal of Single Image Super-Resolution (SISR). Conventional techniques often result in photos with poor textures and artifacts, struggling to produce realistic and detailed images. This work introduces a novel approach that addresses these limitations: Best-Buddy GANs (Beby-GAN).

Best-Buddy GANs tackle this problem through several significant innovations:

- **One-to-Many Best-Buddy Loss:** Beby-GAN moves away from the strict one-to-one mapping typical GANs use between LR and HR images. Instead, it allows each estimated patch to search multiple "best buddies" among HR image patches dynamically during training, providing reliable guidance. This adaptability enhances the creation of more realistic and rich textures.
- **Region-Aware Adversarial Learning:** This technique trains the model to focus on producing details in areas with texture. During training, a region-aware discriminator is incorporated, giving larger weights to errors in textured areas. The resulting images exhibit more realistic and clear details.

Beby-GAN comprises two primary components: a discriminator and a generator.

- **Generator:** The generator, a deep convolutional neural network, reconstructs the HR image from the LR input. The learning process is guided by the best-buddy loss, enabling each estimated patch to seek multiple appropriate HR patches for supervision.
- **Discriminator:** This additional convolutional neural network distinguishes between artificially created and genuine HR images. It integrates the region-aware learning technique to focus on producing realistic details, particularly in textured areas.

The efficacy of Beby-GAN is demonstrated through extensive trials on various datasets and upscaling factors. In comparison to earlier techniques, the method produces visually superior images with richer textures and fewer artifacts, achieving state-of-the-art performance on both quantitative and qualitative parameters. Beby-GAN represents a noteworthy development in SISR research, surpassing current technological limitations by utilizing a unique best-buddy loss and region-aware adversarial learning approach to produce incredibly realistic and detailed images.

2.6 PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION

Creating realistic details and high-resolution images can be a challenge for Generative Adversarial Networks (GANs). Moreover, there is a risk of mode collapse and instability during the training phase. To address these issues, this paper introduces a revolutionary method termed "progressive growing." The main concept is to train the GAN's discriminator and generator incrementally, starting at a low resolution and then adding layers to represent progressively finer details.

The architecture proposed in "Progressive Growing of GANs" fundamentally involves a progressive increase in both the generator and discriminator architectures during training. The process begins with a minimal resolution, typically 4x4 pixels, and systematically introduces additional layers for higher resolutions. This incrementally growing architecture enables the model to learn coarse features initially and progressively refine them, capturing fine details as the resolution increases.

At each stage, a new layer is appended to both the generator and discriminator, facilitating the generation of images with greater clarity and complexity. The generator synthesizes images from lower to higher resolutions, while the discriminator evaluates both real and generated images at varying levels of detail. This multi-resolution training mitigates issues like mode collapse by encouraging the model to learn a diverse set of features at each stage.

The authors utilize skip connections between consecutive layers, allowing information to flow easily across resolutions. Batch normalization and a careful initialization strategy contribute to the stability of the training process. The resulting architecture not only addresses challenges such as mode collapse and instability but also produces high-quality images with intricate details, marking a significant advancement in the training of generative models.

The loss function in "Progressive Growing of GANs" involves a combination of adversarial loss, feature matching loss, and path regularization, ensuring the generator and discriminator are trained effectively. The authors emphasize the importance of carefully balancing these components to achieve stable and high-quality results.

In terms of results, the progressive growing approach demonstrates remarkable success in overcoming challenges associated with traditional GANs. The generated images exhibit superior quality, increased stability, and a diverse range of realistic details. Quantitative evaluations on standard datasets such as CelebA and LSUN consistently show the effectiveness of the proposed method.

In conclusion, "Progressive Growing of GANs" presents a pioneering approach to training generative models by incrementally growing both the generator and discriminator during training. The introduced architecture, loss function, and training strategy collectively address critical issues in GANs, resulting in significant advancements in image generation quality, stability, and diversity. The progressive grow-

ing technique has since become a cornerstone in the development of state-of-the-art generative models, influencing subsequent research in the field.

2.7 Supervised Pixel-Wise GAN for Face Super-Resolution

Face photos with low quality often appear hazy and lack detail, rendering them unsuitable for various purposes. The objective of face super-resolution (SR) is to extract high-resolution (HR) face photographs from their low-resolution counterparts while preserving essential facial features and photorealism. Current SR techniques, including deep learning and conventional interpolation, frequently introduce artifacts and struggle to reliably reconstruct tiny details, especially in crucial facial regions like the lips and eyes. To overcome these drawbacks, Supervised Pixel-Wise GAN (SPGAN) introduces a completely novel Generative Adversarial Network (GAN) designed specifically for face super-resolution. SPGAN employs two significant breakthroughs to achieve cutting-edge results:

- **Supervised Pixel-Wise Discriminator:** SPGAN utilizes a pixel-wise discriminator instead of the more common patch-based or global discriminators seen in normal GANs. This allows for a precise and comprehensive comparison of every pixel in the generated SR image with its corresponding pixel in the ground-truth HR image. Focusing on individual pixel-level photorealism significantly reduces fuzzy artifacts, enabling more accurate evaluation of the produced image quality.

- **Discriminative Matrix and Supervised Pixel-Wise Adversarial Loss:** SPGAN incorporates a discriminative matrix to leverage prior knowledge about facial structures and characteristics, further optimizing the reconstruction process. The generator prioritizes the reconstruction of important facial features based on this matrix, resulting in higher-quality and more realistic generated images. To ensure accuracy and naturalness in the reconstruction, a supervised pixel-wise adversarial loss penalizes the generator for any deviations at the pixel level from the ground-truth image.

The combination of these developments positions SPGAN as a significant improvement over current SR techniques. Superior picture integrity and retention of structural details are indicated by its high peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM). Furthermore, SPGAN excels in recreating challenging features like mouths and eyes, producing faces that appear more realistic and natural.

In summary, SPGAN represents a crucial advancement in face super-resolution. It proves to be an effective tool for various applications in image forensics, video surveillance, and human-computer interaction due to its capacity to generate realistic, high-quality face images, providing faster training, controlled reconstruction, and enhanced face recognition.

2.8 Improving GAN Performance in Single Medical Image Super-Resolution

Introduction

This study addresses the challenges of Single Image Super-Resolution (SISR) in medical images, particularly brain tumor MRI images. Existing GAN-based models face issues such as lower signal-to-noise ratios and synthesizing unrealistic patterns. To overcome these challenges, the paper introduces a Lesion Focused SR (LFSR) method, leveraging Wasserstein GAN (WGAN), WGAN with Gradient Penalty (WGAN-GP), and a novel Multi-Scale GAN (MS-GAN) for improved stability and perceptual quality.

The demand for high-resolution medical images, particularly in clinical applications such as brain tumor diagnosis, underscores the importance of Single Image Super-Resolution (SISR). This paper explores the challenges unique to medical images, emphasizing the need for more effective and realistic solutions. Existing SISR methods, especially those based on GANs, are reviewed, highlighting their limitations when applied to medical images. The paper introduces the concept of Lesion Focused SR (LFSR) and proposes a novel Multi-Scale GAN (MS-GAN) to address these challenges.

Methodology

The GAN variations considered in the study, including WGAN and WGAN-GP, are described. The LFSR method, incorporating a lesion detection neural network and a deep residual super-resolution neural network, is detailed. Additionally, the MS-GAN architecture is introduced, designed to generate multi-scale SR images for more challenging tasks. The experiments utilize the BraTS 2018 datasets, comprising MRI images from brain tumor patients. The proposed methods are compared against existing GAN variations, and various evaluation metrics, including Peak Signal-to-Noise Ratio (PSNR), Structural SIMilarity (SSIM), and a Mean Opinion Score (MOS), are employed. Quantitative results demonstrate that MS-GAN outperforms other GAN variations in terms of MOS, indicating improved perceptual quality. Visualizations and comparisons with other methods showcase the effectiveness of LFSR coupled with MS-GAN in preserving lesion characteristics and enhancing image quality.

Conclusion

The study concludes by highlighting the threefold contributions of LFSR and MS-GAN: lesion-focused attention, exploration of advanced GAN architectures, and the development of a multi-scale model. The proposed approach shows promise for spatial resolution enhancement in brain tumor MRI images and holds potential for broader clinical applications.

2.9 To learn image super-resolution, use a GAN to learn how to do image degradation first

The main focus of this paper is on the problem of super-resolving real-world low-resolution images for a specific object category. The paper uses faces as its "object", but the method is potentially applicable to other objects.

The major contribution of this technique is about how the dataset was generated: while the other techniques use as input low-resolution images which are artificially generated by simple bilinear down-sampling or in a few cases by blurring followed by down-sampling, this paper presents one of the very first attempts towards real-world image super-resolution.

The system that this paper suggests constitute of a GAN, called Low-to-High, that takes 16×16 images and outputs 64×64 images. The main difference between this paper and the others is the way the training dataset was generated: In most prior work, the LR images are produced by bilinearly downsampling the corresponding (original) HR images, which completely ignores the degradation process (e.g. motion blur, compression artefacts etc.). To alleviate this, the paper proposes to learn both degrading and downsampling a HR facial image using another network which is called High-to-Low.

Both the generator and the discriminator of the High-to-Low GAN are based on ResNet architectures. The generator of this GAN takes as input the HR image concatenated with a noise vector that was projected and then reshaped using a fully connected layer in order to have the same size as one image channel. On the other hand, the discriminator takes the generated image as input and tries to detect if it came from the same LR distribution as the real images. The total loss of this GAN is a combination of a GAN loss and an L_2 pixel loss. The model uses L_2 pixel loss with a small coefficient to enforce that the output of the generator has similar content

The generator of the Low-to-High GAN accepts as input the output of the High-to-Low network and tries to generate the original HR image. The discriminator is the same as the one used in High-to-Low, with the exception that it takes as input the generated HR image and tries to detect if that it is fake.

The method was compared with SRGAN, Wavelet-SRNet, FSRNet, CycleGan, and DeepDeblur. The method outperforms all other variants and methods considered both numerically (in terms of FID with a score of 14.89) and (more importantly) visually.

2.10 Multi-Perspective Discriminators-Based Generative Adversarial Network for Image Super Resolution

Super-resolution (SR) picture creation challenges have been transformed by Generative Adversarial Networks (GANs). Nevertheless, current GAN-based SR techniques usually encounter difficulties in generating high-quality images, sometimes exhibiting aberrations such as irregular patterns and synthetic textures. These artifacts impede their practical usage and severely deteriorate visual quality.

In order to successfully address these issues, this work suggests a novel Multi-Perspective Discriminators-Based SR-GAN architecture. The network learns more thorough representations of the target high-resolution (HR) images by employing several discriminators that concentrate on different facets of image quality, which results in artifact-free SR reconstruction.

The capacity of GAN-based SR techniques to produce realistic and artifact-free images is essential to their success. Nevertheless, existing approaches frequently fail, presenting a number of difficulties: First, Pieces of grid: a grid-like pattern that develops in the output image as a result of the network's poor handling of high-frequency information. Second, Unpleasant high-frequency (HF) components: The resulting image has unnatural and noisy textures, which lowers the image's visual quality even more. The actual usability of GAN-based SR techniques is limited by these distortions, particularly in applications where great image quality is required.

Several discriminators are used in the suggested model, each of which focuses on a distinct facet of the generated image quality. By giving the generator more direction, these discriminators enable the generator to learn more complete representations and efficiently suppress artifacts.

The DCT perspective discriminator is one important discriminator. In order to locate and eliminate checkerboard artifacts, it examines the frequency domain of the created image. This is accomplished by contrasting the frequency spectrum of the produced image and the ground-truth HR image.

The gradient perspective discriminator is another significant discriminator. It locates and eliminates undesirable HF components by concentrating on the gradient magnitude distribution of the created image. This discriminator directs the network to generate textures that are more realistic and natural-looking by comparing the gradient magnitude distribution of the generated image with that of the HR image.

These multi-perspective discriminators are included into the suggested SR-GAN architecture, which enables it to learn to produce SR images of excellent quality that are visually pleasing and devoid of artifacts.

To sum up, compared to existing methods, the proposed Multi-Perspective SR-GAN achieves superior performance, generating high-quality SR images that are closer to the original ones and more visually appealing. This is objectively demonstrated by lower PSNR and SSIM scores and subjectively confirmed by higher PI and MOS scores.

2.11 ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

ESRGAN is a super-resolution AI model that is considered an enhancement of the SRGAN model. ESRGAN is perceptual-loss-oriented because PSNR-oriented approaches tend to output over-smoothed results without sufficient high-frequency details, since the PSNR metric fundamentally disagrees with the subjective evaluation of human observers. On the other hand perceptual loss optimize super-resolution model in a feature space instead of pixel space, which makes the output sharper.
The architecture of Generator ESRGAN is the same as that of SRGAN with 2 modifications:

1. Removing all Batch Normalization(BN) layers, which has proven to increase performance and reduce com- putational complexity in different PSNR-oriented tasks including SR and deblurring. This also helps to improve generalization ability and to reduce computational complexity and memory usage.

2. Replacing the original basic block with the proposed Residual-in-Residual Dense Block (RRDB). The proposed RRDB employs a deeper and more complex structure than the original residual block in SRGAN.

On the other hand the paper changes the discriminator of SRGAN and uses instead a relativistic discriminator, which tries to predict the probability that a real image x_r is relatively more realistic than a fake one x_f. more precisely, the discriminator takes as input two images and tries predict wheather the first image is more realistic than the second image. So, the discriminator loss becomes:

$$L_D^{Ra} =_{x_r} [\log(D_{Ra}(x_r, x_f))] -_{x_f} [\log(1 - D_{Ra}(x_r, x_r))]$$

where x_f is the generated image, and x_r is the real image. And the generator loss is:

$$L_G^{Ra} =_{x_r} [\log(1 - D_{Ra}(x_r, x_r))] -_{x_f} [\log(D_{Ra}(x_r, x_f))]$$

The paper adds perceptual loss that perserves the content of the image. Perceptual loss is previously defined on the activation layers of a pretrained deep network, where the distance between two activated features is minimized. Contrary to the convention, the author proposes to use features before the activation layers, which gives more pleasing results. The total Generator loss is:

$$L_G = L_{percep} + \lambda L_G^{Ra} + \eta L_1$$

ESRGAN results were compared with PSNR-oriented methods including SRCNN], EDSR, and RCAN,

and also with perceptual-driven approaches including SRGAN and EnhanceNet. ESRGAN outperforms previous approaches in both sharpness and details.

2.12 Blind Super-Resolution Kernel Estimation using an Internal-GAN

The basic model of SR assumes that the low-resolution input image I_{LR} is the result of down-scaling a high-resolution image I_{HR} by a scaling factor s using some kernel k_s. Real LR images, however, rarely obey this assumption. The aim of this paper not to build a super-resolution model, but to build a model that takes as input the LR images and find its underlying image-specific SR kernel. More generally, the paper tries to build a model that can learn the LR distribution and produce new LR images from existing HR images.

The model architecture consists of a downscaling generator (G) and a discriminator (D), with both of them being fully-convolutional. Given an input image I_{LR}, G learns to downscale it, such that, for D, at the patch level, it is indistinguishable from the input image I_{LR}. On the other hand, D trains to output a heat map, referred to as D-map indicating for each pixel, how likely is its surrounding patch to be drawn from the original patch-distribution. This map is then compared with the ground truth (ones for real data and zeros for fake data) wit MSE. Once converged, the generator G^* constitutes, implicitly, the ideal SR downscaling function for the specific LR image.

Since downscaling by convolution and subsampling is a linear transform, the authors use a linear Generator (without any non-linear activations) because without an explicit constraint on G to be linear, it will generate physically unwanted solutions for the optimization objective. Such solutions include generating any image that contains valid patches but has no downscaling relation.

After training, the model learns the LR distribution, but it is better to extract the kernel from the network for 2 reasons:

1. we are interested in having a compact downscaling convolution kernel rather than a downscaling network. Once extracted, the kernel can be combined with an existing SR model.

2. allow applying explicit and physically-meaningful priors on the kernel.

Because the network is fully convolutional, extracting the kernel is feasible by convolving all the filters of G sequentially with sride 1.

For evaluation of the kernel estimation the author choses two non-blind SR algorithms that accept a SR-kernel as input, provide them with different kernels (bicubic, ground truth SR-kernel, and this) and compared their SR performance. More specifically, this model was compare with a similar one (Michaeli & Irani) combined with SRMD and ZSSR. This method outperformed Michaeli & Irani in both PSNR and SSIM metrics.

2.13 Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data

Most super-resolution approaches assume an ideal bicubic downsampling kernel, which is different from real degradations. This degradation mismatch makes those approaches unpractical in real-world scenarios. In this paper, the author aim to extend the powerful ESR- GAN to restore general real-world LR images by synthesizing training pairs with a more practical degradation process: the model consists of a high order degradation model combined with sinc filters to remove common ringing and overshoot artifacts.

The classical degradation process (which is the base step in the high order degradation model) consists of four parts:

1. **Blur**: Blur degradation is modeled as a convolution with a linear blur filter (kernel).
2. **Noise**: Two commonly-used noise types are used in the model: Gaussian noise and Poisson noise.
3. **Downsampling**: Several resize algorithms - nearest-neighbor interpolation, area resize, bilinear interpolation, and bicubic interpolation- are employed in the model.
4. **JPEG compression**: It first converts images into the YCbCr color space and downsamples the chroma channels. Images are then split into 8×8 blocks and each block is transformed with a two-dimensional discrete cosine transform (DCT), followed by a quantization of DCT coefficients.

The classical degradation model is still not enough for mimicking the real LR distribution, so the author employs a high-ordered degradation model, which involves several classical degradation processes with the same procedure but different hyper-parameters. Further more a sinc kernel is added to the model to synthesize ringing and overshoot artifacts for training pairs.

For the SR model architecture, the paper uses the same ESRGAN generator architecture, but changes the VGG-style discriminator to a U-Net discriminator. To stabilize the training dynamics, the spectral normalization regularization is used.

Real-ESRGAN was comapred with several state- of-the-art methods, including ESRGAN, DAN, CDC, RealSR, and BSRGAN. Real-ESRGAN outperforms previous approaches in both removing artifacts and restoring texture details. However Real-ESRGAN has some limitations:

1. some restored images (especially building and indoor scenes) have twisted lines due to aliasing issues
2. GAN training introduces unpleasant artifacts on some sam- ples
3. It could not remove out-of-distribution complicated degradations in the real world, and may make it worse by amplifying the artifacts.

2.14 R-SRGAN: Residual Super-Resolution Generative Adversarial Networks

Abstract

Super-Resolution (SR) techniques, reconstructing high-resolution images from low-resolution counterparts, have widespread applications in fields like object detection, medical imaging, and remote sensing. This paper presents an image super-resolution method based on R-SRGAN (Residual Super-Resolution Generative Adversarial Networks) to address issues in GAN training such as instability and mode collapse. The proposed method employs residual blocks to retain detailed information and utilizes Wasserstein distance as a loss function for enhanced training effects.

Introduction

With the growing importance of images in information transmission, the need for high-resolution images is crucial. Image Super-Resolution (SR) methods have emerged to convert low-resolution images into high-resolution ones. Traditional methods involve interpolation, but recent advancements in deep learning, such as SRCNN and SRGAN, have shown promise. This paper introduces R-SRGAN as an improvement, leveraging Generative Adversarial Networks (GANs) for better super-resolution performance.

Related Work

The history of image super-resolution dates back to 1964, with early methods focusing on combining prior knowledge to regain high-frequency information. Traditional methods often involve interpolation, but recent developments in deep learning, including SRCNN and SRGAN, have demonstrated significant improvements. The paper discusses the principles of GANs, introduces the Wasserstein GAN (WGAN) to address stability issues, and outlines the objective function of GANs.

Residual Super-Resolution Generative Adversarial Networks (R-SRGAN)

The R-SRGAN model consists of a generator (G) and a discriminator (D). Residual blocks are added to the generator's structure, enhancing the human eye's perception of generated images and stabilizing the deep network. The discriminator adopts the VGG structure. The loss function includes Mean Squared Error (MSE) loss and adversarial loss, utilizing the EarthMover (EM) distance for improved diversity in generated samples.

Experiment

The DIV2K dataset, containing 1000 2K resolution images, is used for training, validation, and testing. The paper evaluates the proposed R-SRGAN method against benchmark datasets (Set5, Set14, BSD100) using Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM). Results show that R-SRGAN outperforms bicubic interpolation, SRCNN, and SRGAN in terms of PSNR and SSIM.

Conclusion

The paper concludes that R-SRGAN is effective in image super-resolution, providing better results than traditional interpolation and previous deep learning methods. The method demonstrates improved image contrast and line sharpness, emphasizing its practical applicability. However, the paper acknowledges some limitations, such as early-stage color instability and insufficient detail recovery, which warrant further investigation.

Chapter 3

Computer Vision Papers

3.1 Simple Online and Realtime Tracking with a Deep Association Metric

In the paper "Simple Online and Realtime Tracking with a Deep Association Metric," the authors introduce an extension to the Simple Online and Realtime Tracking (SORT) algorithm. The primary objective is to enhance tracking performance by integrating appearance information, reducing identity switches, and improving object tracking through occlusions.

To achieve this, a deep association metric trained on a large-scale person re-identification dataset is introduced. This metric allows the algorithm to establish associations between measurements and tracks by leveraging nearest neighbor queries in visual appearance space. By incorporating this deep association metric, SORT becomes capable of better handling occlusions and reducing the number of identity switches.

One performance metric equation that was introduced is Mahalanobis distance, which serves as an association metric. It quantifies the dissimilarity between feature vectors and aids in making informed decisions for accurate tracking. The Mahalanobis distance equation is defined as follows:

$$D^2 = (x - \mu)^T \Sigma^{-1} (x - \mu)$$

In this equation:

- D represents the Mahalanobis distance.
- x is the feature vector of a measurement or track.
- μ is the mean feature vector of a track.
- Σ is the covariance matrix of the track's features.

The Mahalanobis distance takes into account the statistical properties of the track's features and measures the dissimilarity between the feature vectors. By calculating and comparing the Mahalanobis distance, the algorithm can make informed decisions about associations between measurements and existing tracks, enabling effective tracking even in complex scenarios.

The experimental evaluation of the extended SORT algorithm demonstrates its effectiveness. The integration of the deep association metric leads to a significant reduction of 45% in the number of identity switches. This reduction indicates improved tracking consistency, particularly in challenging situations involving occlusions. Additionally, the extended SORT algorithm maintains competitive performance at high frame rates, making it suitable for real-time tracking applications.

The paper also highlights the importance of considering certain edge cases. For example, unaccounted camera motion can introduce rapid displacements in the image plane, affecting tracking through occlusions. To mitigate such issues, the authors suggest thresholding the Mahalanobis distance at a 95% confidence interval, excluding unlikely associations and improving overall tracking accuracy.

In conclusion, the paper presents an extension to the SORT algorithm by integrating appearance information through a deep association metric. The Mahalanobis distance equation quantifies dissimilarity, aiding accurate tracking decisions. The extended SORT algorithm demonstrates a significant reduction in identity switches and achieves competitive performance at high frame rates. Considerations for unaccounted camera motion, thresholding the Mahalanobis distance, and object detector performance provide valuable insights for multiple objects tracking applications.

3.2 StrongSORT: Enhancing Multi-Object Tracking with AFLink and GSI

The paper introduces StrongSORT, a baseline for multi-object tracking (MOT) that enhances the classic tracker DeepSORT. The authors address challenges in MOT, including object detection, feature embedding, trajectory association, and missing association and detection. They propose two lightweight algorithms, AFLink and GSI, to handle missing associations and missing detections, respectively. StrongSORT, along with AFLink and GSI, achieves state-of-the-art results on public benchmarks such as MOT17, MOT20, DanceTrack, and KITTI.

In MOT, the tracking-by-detection (TBD) paradigm is widely used, where per-frame detection and data association are performed. DeepSORT, an early deep learning-based method, serves as the basis for StrongSORT. The authors improve DeepSORT by incorporating a strong detector and embedding model, along with inference tricks from recent works. This results in the proposed StrongSORT baseline, which outperforms state-of-the-art methods on MOT17 and MOT20 benchmarks.

To address missing association, where the same object is spread across multiple tracklets, the paper introduces AFLink. Unlike previous methods that rely on appearance embeddings, AFLink performs global association without appearance information. It achieves a balance between speed and accuracy by associating short tracklets into complete trajectories at a lower computational cost.

For the missing detection problem, the authors propose GSI. This algorithm uses Gaussian process regression and Gaussian-smoothed interpolation to compensate for missing detections caused by occlusion and low resolutions. GSI effectively generates interpolated bounding boxes based on the existing detections, improving tracking performance as follows:

$$X = \sum_{i=1}^{N} (w_i \cdot X_i)$$

In this equation:

- X represents the interpolated bounding box to be estimated,
- X_i represents the existing detections,
- w_i represents the weight assigned to each detection,
- N represents the total number of existing detections.

The weights w_i are determined using Gaussian process regression, which takes into account the spatial and temporal relationships between the existing detections. By using this equation, GSI effectively estimates the interpolated bounding box X based on the available detections, improving the accuracy of the tracking results.

3.3 Towards Real-Time Multi-Object Tracking

The paper introduces a real-time multi-object tracking (MOT) system that aims to improve the efficiency of object tracking. Traditional MOT systems follow a two-step approach, involving target detection and data association. However, executing these steps separately can lead to efficiency issues. Existing real-time MOT methods mainly focus on the association step, neglecting the overall system's real-time capability.

To address this, the paper proposes a novel MOT system where target detection and appearance embedding are learned in a shared model. This integration allows the model to simultaneously output detections and corresponding embeddings, reducing computation costs significantly. Additionally, a simple and fast association method is introduced to work in conjunction with the joint model.

The proposed system achieves near real-time performance, with a running speed of 22 to 40 frames per second (FPS) depending on the input resolution. Notably, this is the first (near) real-time MOT system reported in the literature. Despite its real-time capability, the tracking accuracy of the proposed system remains comparable to state-of-the-art trackers that employ separate detection and embedding learning.

One of the key contributions of the paper is the introduction of the Joint Detection and Embedding (JDE) model. This model adopts a single-shot deep network that simultaneously outputs detection results and appearance embeddings. Unlike traditional methods that rely on resampled pixels or feature maps, the JDE model shares low-level features, avoiding unnecessary recomputation. This approach significantly improves efficiency while maintaining accuracy.

The authors conducted thorough analysis and experiments to develop the JDE model. They explored various aspects, including training data, network architecture, learning objectives, optimization strategies, and validation metrics. The experiments demonstrated that the JDE model performs as well as strong combinations of separate detection and embedding models while achieving the fastest speed.

In conclusion, the "Towards Real-Time Multi-Object Tracking" paper introduces a real-time MOT system that integrates target detection and appearance embedding learning. The proposed Joint Detection and Embedding (JDE) model achieves near real-time performance and comparable tracking accuracy to state-of-the-art methods. This work provides new insights and baselines for joint detection and embedding learning, contributing to the advancement of real-time MOT research.

3.4 YOLO (You Only Look Once)

Introduction

The paper presents YOLO, an abbreviation for "You Only Look Once." YOLO is an object detection method designed to achieve real-time performance by effectively addressing the trade-off between speed and accuracy. The importance of efficient object detection systems in various applications is emphasized, highlighting the need for faster processing without compromising detection accuracy.

Methodology

YOLO introduces a unified architecture that divides the image into a grid and performs object detection by regressing bounding boxes and class probabilities simultaneously. This innovative approach enables real-time detection by eliminating the need for multiple passes through the image.

Results

The YOLO model demonstrates real-time object detection capabilities while maintaining high accuracy compared to previous methods. It successfully achieves an efficient trade-off between speed and accuracy, making it suitable for various real-world applications.

Discussion

The findings suggest that the unified architecture of YOLO significantly improves object detection speed without compromising accuracy. However, challenges exist in detecting small objects due to constraints imposed by the grid cell size.

Conclusion

The YOLO paper introduces an innovative approach to object detection, revolutionizing real-time detection capabilities. By addressing the trade-off between speed and accuracy, YOLO lays the groundwork for further improvements in the field.

3.5 Machine-learning-based Top-View Safety Monitoring of Ground Workforce on Complex Industrial Sites

The study addresses safety concerns in crane operations, introducing CraneNet—a real-time machine learning-based human detection system for top-view perspectives in cluttered industrial sites. Deployed on the compact Jetson AGX Xavier device, CraneNet employs a specialized CNN to detect small-sized human objects via a video camera on the crane hook. Contributions include a power-efficient top-view object detection system design, a novel CNN for crane-specific human detection, and CraneNet's training/validation. Real-world application at an industrial site in Stirling validates its efficacy, providing a high-accuracy, efficient solution for real-time human detection, enhancing safety in challenging industrial environments.

The proposed CraneNet system employs a small-form PC to process video streams from a crane-mounted camera, utilizing a modified ResBlock-D and a PAN module for real-time, top-view human detection in industrial settings. The system enhances object detection speed and accuracy, crucial for small human objects at high altitudes. CraneNet's architecture includes a backbone with modified ResBlock-D modules, Spatial Pyramid Pooling, and a PAN module to improve small object detection. The PAN module reduces the data path between layers, optimizing feature combinations. The integration of these components results in an efficient and accurate solution for real-time human detection in challenging industrial environments, particularly during crane operations.

To optimize real-time execution of the proposed CraneNet, a Multi-Thread Pipeline software divides the execution platform into main and detection threads. The main thread manages video frames, deciding whether to process based on detection thread availability. A blocking queue system facilitates coordination. The detection thread executes the object detection algorithm on frames, involving pre-processing and providing results such as object class, location, and confidence score, addressing the computational challenges of convolutional neural networks for efficient real-time operation in crane-mounted cameras.

The proposed object detection system, CraneNet, was deployed in an industrial site in Stirling, Scotland, using a Reolink camera on crane hooks. The system, embedded on an NVIDIA Jetson Xavier, achieved high accuracy (92.59%) and 19 FPS, surpassing other state-of-the-art models like YOLOv4 and Tiny-YOLOv4 in both speed and accuracy for top-view small human detection, crucial for workforce safety in crane operations. The evaluation demonstrates CraneNet's effectiveness in a real-world scenario, offering a balance between accuracy and speed.

The study's findings highlight CraneNet's superior performance in real-world industrial scenarios for top-view human detection. While surpassing existing models, the limitations include potential challenges in extremely cluttered environments. Despite this, the research underscores significant implications for improving safety in crane operations through accurate and efficient human detection.

In conclusion, CraneNet, a real-time top-view human detection system, demonstrated superior accuracy (92.59%) and efficiency (19 FPS) in industrial settings. Addressing safety concerns in crane operations, CraneNet outperforms existing models, showcasing its potential for widespread application. Future studies can build on this research, advancing safety measures in dynamic industrial environments and contributing to the ongoing development of efficient, real-time machine learning-based systems.

3.6 YOLO-based Human Action Recognition and Localization

Introduction

The paper addresses the need for efficient human action recognition in video analytics, emphasizing the disparity between traditional methods and human vision strategies. The research aims to leverage the You Only Look Once (YOLO) approach for real-time detection, localization, and recognition of human actions, presenting a valuable contribution to video analytics applications.

Methodology

The study employs YOLO for its effectiveness and speed, utilizing frames from the LIRIS Human Activities dataset. The model is designed to process frames at specified intervals, providing action labels based on minimal visual data. Training involves converting LIRIS labels and configuring YOLO parameters, with subsequent testing demonstrating the model's real-time capabilities on standard desktop PCs.

Results

The proposed model achieves an 88.37% accuracy on the LIRIS test dataset, surpassing certain state-of-the-art methods in terms of F-scores. The YOLO-based approach demonstrates its efficacy in recognizing and localizing human actions, presenting a promising solution for video analytics applications.

Discussion

Interpreting the findings in the context of existing literature, the paper acknowledges the limitations of current approaches and highlights the efficiency of YOLO. The discussion emphasizes the potential applications in surveillance, robotics, and human-computer interfaces, paving the way for future studies to explore enhanced action recognition through object detection and additional metrics.

Conclusion

In conclusion, the paper underscores the effectiveness of YOLO for real-time human action recognition, emphasizing its potential in various domains. The research question regarding efficient recognition with minimal frames is successfully answered, opening avenues for future studies to build upon these findings for improved video analytics applications. The paper's significance lies in its practical implementation and promising results.

Additional Tips

The paper aligns with a theoretical framework that emphasizes real-time action recognition. Ethical considerations are implicit, focusing on video analytics applications. While funding sources are not explicitly mentioned, the study's practical implications for surveillance and robotics suggest potential industry applications.

3.7 YOLO ”YOLO9000” (You Only Look Once)

Introduction

Building upon the success of YOLO, this paper aims to further enhance object detection performance by increasing the number of detectable object categories.

Methodology

YOLO9000 introduces improvements in the training process and data augmentation techniques to expand the model's capability to detect a more extensive range of object categories. It utilizes a joint training approach to leverage both labeled and unlabeled data.

Results

The YOLO9000 model achieves superior performance in detecting a significantly increased number of object categories, surpassing the limitations of previous versions. It demonstrates the scalability and adaptability of the YOLO framework.

Discussion

The paper discusses the challenges of scaling object detection models to thousands of categories and the importance of leveraging hierarchical label taxonomies. It addresses the trade-offs between model complexity, speed, and detection accuracy.

Conclusion

YOLO9000 advances object detection by significantly increasing the number of detectable object categories while maintaining the real-time detection capabilities introduced in the original YOLO paper. It lays the groundwork for scalable and comprehensive object detection systems.

3.8 A Review of YOLO Algorithm Developments

Introduction

The paper explores the evolution of the You Only Look Once (YOLO) algorithm and its subsequent versions, highlighting the algorithm's significance in the artificial intelligence field. The research aims to analyze the differences and similarities among YOLO versions, focusing on the ongoing improvement of the algorithm.

Methodology

The study employs a comparative approach, analyzing the conceptual designs and implementations of five main YOLO versions: YOLO, YOLO V2, YOLO V3, YOLO V4, and YOLO V5. Methodologically, the paper reviews the key features, developments, and improvements introduced in each version. The analysis involves trends and queries collected from public data sources such as Google and YouTube.

Results

The main findings emphasize the continuous improvement of the YOLO algorithm, with each version introducing novel features and addressing limitations. Key improvements include batch normalization, high-resolution classifiers, fine features, and multi-scale training. YOLO V4, in particular, integrates various methods, achieving high performance and efficiency.

Discussion

Interpreting the findings in the context of existing literature, the paper discusses the core characteristics of YOLO, its strengths, and limitations. It addresses challenges such as inaccurate positioning and lower recall rates, providing insights into the modifications introduced in each version to overcome these limitations. Ethical considerations and potential applications are briefly touched upon.

Conclusion

In conclusion, the paper synthesizes the main points of the YOLO algorithm's evolution, highlighting the ongoing nature of its development. The research question regarding the improvement of YOLO is addressed through a comprehensive analysis of its versions. The significance of the research for future studies and applications is underscored, emphasizing the need for continued exploration and improvement in object detection techniques.

3.9 A Novel Real-Time Multiple Objects Detection and Tracking Framework for Different Challenges

Introduction

This paper introduces a novel algorithm addressing challenges in real-time object detection and tracking, crucial in security and surveillance. The proposed method combines principal component analysis (PCA) and deep learning networks adaptively for enhanced performance. It aims to provide responsive and effective detection and tracking of objects in natural scenes captured by real-time cameras.

The escalating use of surveillance cameras prompts the need for efficient real-time solutions in detection and tracking. This algorithm caters to challenges faced by monitoring systems, offering a responsive and effective solution applicable to various scenarios such as early detection of alien organisms or violent crime. A review of existing monitoring systems emphasizes the significance of object detection and tracking steps. Challenges, such as changes in lighting and camera angles, impacting system performance, are discussed. The proposed algorithm addresses gaps identified in previous algorithms and systems.

Methodology

The paper explores the fundamental principles of the proposed algorithm, integrating PCA for dimension reduction and background modeling with deep learning networks for object detection. Challenges associated with training deep learning networks and the impact of factors like camera type on detection accuracy are addressed. The structure of the proposed system is detailed, outlining stages for object detection and tracking. The object detection stage incorporates modern techniques, including single-stage and two-stage detectors, highlighting the trade-off between speed and accuracy. The tracking stage employs a Kalman filter and adaptive scales for precise tracking. The proposed system's performance is evaluated using diverse datasets, including highway data and standard datasets like Change Datasets 2014 (CD) and MOT16. Metrics such as Multiple Objects Tracking Accuracy (MOTA) and Multiple Objects Tracking Precision (MOTP) validate the algorithm's efficiency.

Results and Comparative Analyses

Comparative analyses with deep learning and PCA-based methods demonstrate the proposed algorithm's superiority.

Conclusion

The paper concludes by emphasizing the adaptability of the proposed algorithm to real-world challenges, offering a practical solution for object detection and tracking in surveillance applications. Future research directions are suggested, including the exploration of additional methods for automatic feature identification and broader applications in crowded or extreme contrast scenes.

3.10 You Only Look Once: Unified, Real-Time Object Detection

Introduction

The paper introduces YOLO (You Only Look Once), a pioneering approach to object detection that fundamentally shifts the paradigm. Unlike conventional methods, YOLO reframes the object detection task as a regression problem, enabling the direct prediction of bounding boxes and class probabilities from complete images in a single evaluation. The beauty of YOLO lies in its unified architecture, facilitating end-to-end optimization for exceptional speed and efficiency. The base YOLO model achieves real-time image processing at an impressive 45 frames per second, establishing its prowess in both rapidity and effectiveness.

One of YOLO's standout features is its global contextual understanding during both training and testing phases. This capability allows YOLO to generalize effectively across diverse domains, showcasing superior performance even in challenging scenarios like artwork detection. Despite occasional localization errors, YOLO exhibits a remarkable ability to minimize false positives, particularly on background elements, setting it apart from its counterparts.

Technical Aspects

The paper provides an in-depth exploration of YOLO's technical aspects, delving into network design, training methodologies, and the inference process. Acknowledging certain limitations, such as spatial constraints on bounding box predictions and challenges associated with small objects grouped together, the paper underscores YOLO's primary source of error—incorrect localizations. Comparisons with existing real-time systems underscore YOLO's dominance in terms of both speed and accuracy.

Hybrid Approach

The paper takes a step further by exploring a hybrid approach, amalgamating YOLO with Fast R-CNN. This integration proves to be highly effective, significantly reducing background mistakes and enhancing overall detection performance. The evaluation extends to the VOC 2012 dataset, where YOLO achieves a mean average precision (mAP) of 57.9%. Notably, a combined Fast R-CNN + YOLO model emerges as one of the highest-performing detection methods, surpassing the standalone Fast R-CNN.

Conclusion

In conclusion, the paper underscores YOLO's versatility, simplicity, and robustness, positioning it as an ideal choice for real-world applications requiring swift and reliable object detection capabilities. YOLO's innovative approach to framing the detection problem as a regression task, coupled with its impressive real-time processing speed, makes it a standout solution in the ever-evolving field of computer vision.

3.11 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision

The paper "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision" addresses the need for lightweight neural network architectures suitable for deployment on mobile and embedded devices. Authored by Andrew G. Howard and his colleagues, the paper was presented at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) in 2017.

The motivation for MobileNets stems from the increasing demand for real-time computer vision applications on resource-constrained devices such as smartphones, IoT devices, and other edge devices. Traditional deep neural networks designed for high accuracy in large-scale settings are often impractical in these scenarios due to their computational complexity and memory requirements.

MobileNets introduce a novel architecture that significantly reduces the computational cost while maintaining competitive accuracy for various vision tasks. The key innovation lies in the use of depthwise separable convolutions, which break down the standard convolutional layer into two distinct operations: depthwise convolution and pointwise convolution.

In the depthwise convolution step, a single filter is applied independently to each input channel. This step captures spatial information within each channel without inter-channel mixing. Following the depthwise convolution, the pointwise convolution combines the outputs of the depthwise step through a 1x1 convolution, allowing cross-channel interactions.

The separation of spatial and channel-wise operations in depthwise separable convolutions drastically reduces the number of parameters and computations compared to traditional convolutional layers. This design choice is crucial for achieving efficiency in terms of model size and inference speed.

The MobileNet architecture is flexible and can be easily customized for different resource constraints by adjusting hyperparameters such as the width multiplier. The width multiplier scales the number of channels in each layer, enabling a trade-off between model size and accuracy. This adaptability makes MobileNets suitable for a wide range of deployment scenarios.

The authors conducted extensive experiments to evaluate the performance of MobileNets on image classification tasks using the ImageNet dataset. The results demonstrated that MobileNets achieved competitive accuracy with state-of-the-art models while being significantly more efficient in terms of model size and inference speed.

In conclusion, MobileNets provide a groundbreaking solution to the challenges of deploying neural networks on mobile and embedded devices. Through the introduction of depthwise separable convolutions, the model achieves a remarkable balance between accuracy and computational efficiency, making it a pivotal contribution to the field of mobile vision applications.

3.12 Introduction to ChainerCV: A Computer Vision Software Library

Abstract

This paper introduces ChainerCV, a computer vision software library designed to support various neural network models and essential software components for conducting research in computer vision. The library aims to provide complete implementations of vision algorithms, emphasizing usability and predictable performance. ChainerCV supports object detection and semantic segmentation tasks, making it suitable for integration into larger software projects, including robotic software systems.

Object Detection and Semantic Segmentation

ChainerCV supports object detection networks, implementing unified approaches using convolutional neural networks for feature extraction and sliding windows for localization and classification. Object detection architectures include Faster R-CNN and Single Shot Multibox Detector (SSD) meta-architectures. Faster R-CNN utilizes external Region Proposal Networks, while SSD directly predicts classes and coordinates of bounding boxes to save processing time.

Semantic segmentation models, including SegNet, follow an encoder-decoder style. A reusable loss calculation module is used, making the loss adaptable to other semantic segmentation implementations.

Dataset Interface

ChainerCV provides an interface to commonly used computer vision datasets such as those from the Pascal VOC Challenge. The dataset object downloads data if necessary, returning contents with an array-like interface. A transform function modifies inputs like image resizing, creating a custom data preprocessing pipeline. The TransformDataset class applies a function to a sample retrieved from the underlying dataset.

Visualization and Evaluation

ChainerCV supports functions for visualization and evaluation crucial for computer vision research. A consistent data representation is enforced across different models. Functions can be used universally, assuming image arrays are RGB and shaped as (C, H, W), where C is channel size, and H and W are height and width of the image.

GPU and CUDA Support

ChainerCV uses `cupy.ndarray` for GPU memory and `numpy.ndarray` for CPU memory. CuPy 2, a GPU-accelerated multidimensional array library similar to NumPy, is employed. Many functions in ChainerCV support both types, with functions like non-maximum suppression efficiently implemented using CUDA parallelization. Installation simplicity is achieved by relying on CuPy for calling CUDA kernels.

3.13 Literature Review: Real-time Object Detection and Tracking Techniques

Abstract

This literature review evaluates five real-time object detection and tracking techniques based on accuracy, computational time, and memory consumption. The objective is to identify the most suitable technique for real-time implementation in mobile robots. The analyzed techniques include: (1) Object tracking by image differencing, (2) Object tracking using non-parametric local transformations, (3) Object tracking using morphological-based object detection, (4) Object tracking using Kanade Lucas technique, and (5) Meanshift algorithm. The results indicate that the Kanade Lucas algorithm is the most feasible for real-time implementation in mobile robots, achieving the best balance of computational time, accuracy, and memory consumption.

Applications

The literature proposes various applications for the identified real-time object detection and tracking technique. Examples include detecting and tracking cars on highways for traffic information, and security surveillance for detecting and tracking suspicious movements. In more complex environments, a combination of traditional and modern methods is suggested, such as background subtraction combined with KLT/meanshift. An interesting application involves outdoor night surveillance, where contrast changes are measured in the first stage, and motion prediction and spatial nearest neighbor data association are employed in the second stage. The proposed system achieves real-time implementation with over 25 FPS at 320×240 resolution using a 1.8G CPU.

Algorithm Performance Comparison

The performance of the analyzed algorithms is summarized in the table below:

Algorithm	**Arithmetic and Logic Operations**	**Time Taken**	**Program Execution Time**
Absolute Differencing	4230100	16	13
Census Transform	2416000	5.4	6
Morphological Tracking	352210	14.2	15
Kanade Lucas	500825	0.486	2

According to the timing analysis, morphological-based tracking is computationally more efficient than Kanade Lucas. However, the issue with morphological-based tracking is that it generates more than

one contour for a single moving object, and its operations increase exponentially with the number of contours. Therefore, Kanade Lucas is considered the most efficient algorithm computationally and the most feasible for real-time implementation. The success of a video tracking algorithm depends on its response to a high frame-rate input video, and Kanade Lucas is recommended for hardware (real-time) implementation in mobile robots due to its speed, low memory consumption, accuracy, and simplicity.

3.14 CVNets: An Open-Source Library for High-Performance Deep Neural Network Training

Abstract

This paper introduces CVNets, a high-performance open-source library for training deep neural networks for visual recognition tasks, including classification, detection, and segmentation. CVNets supports image and video understanding tools, including data loading, data transformations, novel data sampling methods, and implementations of several standard networks. CVNets is a modular system that allows users to integrate various components across various visual recognition tasks for both research and production use cases. It provides independent components, including datasets and models for different tasks and input modalities. CVNets also offers flexibility, allowing easy integration of new components like models, datasets, loss functions, data samplers, and optimizers. Each component is registered in CVNets, allowing for easy use in new use cases. CVNets also provides reproducible implementations of standard models for different computer vision tasks.

Efficient Data Sampling and Training Methods

CVNets include efficient data sampling and training methods, in addition to standard components like optimizers. CVNets offer data samplers with three sampling strategies: (1) single-scale with fixed batch size, (2) multi-scale with fixed batch size, and (3) multi-scale with variable batch size. The single-scale with fixed batch size (SSc-FBS) is the default sampling strategy in deep learning frameworks and libraries like PyTorch, Tensorflow, and MixNet. It samples a batch of b images per GPU with a predefined spatial resolution of height H and width W at the t-th training iteration.

The multi-scale with fixed batch size (MSc-FBS) allows a network to learn representations at multiple scales. It takes a sorted set of n spatial resolutions $S = \{(H_1, W_1), (H_2, W_2), \ldots, (H_n, W_n)\}$ as an input and randomly samples b images per GPU of spatial resolution $(H_t, W_t) \in S$ at the t-th training iteration on each GPU. Multi-scale with variable batch size (MSc-VBS) is more robust to scale changes compared to SSc-FBS. However, it may have higher peak GPU memory utilization depending on the maximum spatial resolution in S, causing out-of-memory errors on GPUs with limited memory. To address this memory issue, MSc-FBS is extended to variably-sized batch sizes. Variably-sized video samplers can be easily extended for videos, allowing control over different video-related input variables for learning space- and time-invariant representations.

Sample Efficient Training (SET)

Sample efficient training: Models learn quickly during the initial phase of training. In other words, models can accurately classify many training data samples during earlier epochs and such samples do not contribute much to learning. Therefore, a natural question arises: Can we remove such samples to reduce total optimization updates? CVNets answer this question with a simple heuristic method, which is called Sample Efficient Training (SET). At each epoch, SET categorizes each sample as either hard or easy. To do so, SET uses a simple heuristic: if the model predicts the training data sample correctly with a confidence greater than a pre-defined threshold τ, then it is an easy sample and we remove it from the training data. At each epoch, the model only trains using hard samples. Because of randomness in training due to data augmentation (e.g., random cropping), it is possible that the region of interest corresponding to the object category may be partially (or not) present in the model's input, and the model may classify such inputs correctly. To make SET robust to such randomness, a training sample is classified easy only when it is classified correctly with a confidence greater than τ for a moving window of w epochs. Because SET uses a window of w epochs to determine easy samples, it is possible that some samples classified easy earlier may be classified harder during the later training stages. SET adds such samples back to the training data.

Chapter 4

Comparison Between SR Models

After examining more than 10 Super-Resolution models, we picked the 3 most convenient models for out application- PGGAN, ESRGAN, and Degradation GAN. The following is a brief comparison between them.

4.1 PGGAN

High-quality image production was greatly aided by the introduction of Progressive Growing GANs (PGGANs), a unique training approach for Generative Adversarial Networks (GANs). PGGANs, which were introduced in 2017 by Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen, tackle issues related to training stability, scalability, and the production of varied and intricate visuals. The main characteristic is the gradual growth of the discriminator and generator networks, which begin at low resolutions and add layers one at a time to handle higher resolutions.

Progressive Growing GANs (PGGANs) have the following benefits when used for super-resolution:

- Training Stability: PGGANs reduce the training instability that GANs are known to exhibit. Stable convergence during training is facilitated by the incremental growth technique, which permits a regulated learning process.
- Quality at higher images: PGGANs are engineered to produce images with superior quality at increasingly greater resolutions. This is in line with super-resolution jobs, where improving image details at larger scales is the main objective.
- Incremental Learning: The model learns slowly, eventually capturing small information according to the progressive development method. This is especially useful for jobs like super-resolution, where producing fine details is crucial.

- Realistic and Diverse Outputs: PGGANs are renowned for creating realistic and perceptually diverse visuals. The model produces visually pleasing outputs because of the progressive training, which enables the model to capture a wide variety of variances in the data distribution.
- Multi-Scale Adaptability: PGGANs are naturally able to operate at a variety of scales, beginning at a lower resolution and gradually increasing it. This feature is useful for jobs like super-resolution that require the creation of multi-scale images.
- Possibility of Faster Convergence: PGGANs may become computationally more efficient in their early training stages if they undergo incremental training from lower to higher resolutions.

In conclusion, Progressive Growing GANs—which provide stability, scalability, and the capacity to produce high-quality, varied images at different resolutions—represent a substantial breakthrough in the training of generative models. Due to these characteristics, PGGANs are an attractive option for jobs such as super-resolution, where generating realistic and detailed high-resolution images is the main objective.

4.2 ESRGAN: Enhanced Super-Resolution Generative Adversarial Network

The ESRGAN is an advanced deep learning model built for single picture super-resolution tasks. It outperforms SRGAN in terms of producing high-quality, realistic-looking textures in super-resolved images.

Key Features of ESRGAN

- **Enhanced Network Architecture:** ESRGAN utilizes the Residual-in-Residual Dense Block (RRDB) without batch normalization as the basic building unit. This architecture is meant to enable it to capture more complex features and textures, thereby making super-resolved images more realistic.
- **Improved Discriminator:** ESRGAN uses RaD, which is based on the Relativistic GAN approach. This discriminator aims at making the model better at distinguishing between real and generated images, leading to sharper edges and more detailed textures in the output.
- **Advanced Perceptual Loss:** For feature comparison, ESRGAN uses a perceptual loss function that operates before activation. This method provides stronger control for brightness consistency and texture recovery, resulting in improved visual quality of super-resolved photographs.

- **Network Interpolation Strategy:** ESRGAN uses a network interpolation strategy to optimize perceptual quality and PSNR. The blur and the amount of noise in pictures are manipulated continuously by this approach to result in better outputs.

Advantages of ESRGAN

- **Improved Visual Quality:** In super-resolution tasks, ESRGAN produces better visual quality with more realistic and natural textures as compared to SRGAN and other GANs. It utilizes complex architectures, a more sophisticated discriminator, and perceptual loss function.
- **Artifact Reduction:**ESRGAN solves the problem of undesirable artifacts that often appear in the super-resolved images generated by other methods. ESRGAN reduces artifacts effectively through the use of advanced techniques and loss functions thus producing clearer and more realistic images.
- **Stable Training:** ESRGAN achieves stable training and consistent performance irrespective of any settings. Batch normalization layers are removed from the model while more improved loss functions are used making sure that it converges stably during training therefore producing reliable results.

In summary, ESRGAN is a revolutionary model for single image super-resolution that provides better visual quality, less artifacts, and more consistent training than existing GAN-based techniques. Its innovative network design, improved discriminator, and perceptual loss function all contribute to its ability to produce high-quality super-resolved images with realistic textures.

4.3 Degradation GAN

The key innovation of this paper lies in its approach to super-resolving real-world low-resolution images, particularly focusing on the specific object category of faces. While previous techniques primarily relied on artificially generated low-resolution images, achieved through simple bilinear down-sampling or occasional blurring followed by down-sampling, this paper stands out by presenting one of the initial attempts at addressing real-world image super-resolution.

The proposed system incorporates a Generative Adversarial Network (GAN) named Low-to-High, designed to take 16×16 images as input and generate high-quality 64×64 images as output. The novel aspect is in how the training dataset is created. Unlike prior methods that merely downsampled high-resolution images without considering degradation processes like motion blur or compression artifacts, this paper introduces a High-to-Low GAN. This network learns both the degradation and downsampling of a high-resolution facial image, providing a more realistic training scenario.

The High-to-Low GAN utilizes ResNet architectures for both the generator and discriminator. The generator takes a high-resolution image concatenated with a noise vector as input, projecting and reshaping it to match the size of one image channel. The discriminator, on the other hand, discerns whether the generated image aligns with the low-resolution distribution of real images. The total loss comprises a combination of GAN loss and L2 pixel loss, with a small coefficient for the latter to ensure content similarity in the generated output.

Comparative evaluation against existing methods, including SRGAN, Wavelet-SRNet, FSRNet, CycleGan, and DeepDeblur, demonstrates the superior performance of the proposed method. Numerical assessments, such as FID with a score of 14.89, along with visual comparisons, substantiate the efficacy of the approach in achieving high-quality super-resolution results.

Chapter 5

Comparison between Computer Vision Models

After thorough examination of diverse computer vision models outlined in numerous research papers, YOLOv8 and MobileNet SSD have emerged as highly promising choices for our project. In the upcoming phase, we will conduct a comparative analysis between these two models to determine their respective strengths.

5.1 MobileNet SSD

Designed for mobile and edge devices, MobileNet SSD (Single Shot Multibox Detector) is a robust and effective object detection method. Its design blends a lightweight object detecting head with the MobileNet architecture for feature extraction, making it ideal for real-time applications with constrained processing power.

5.1.1 Architecture

MobileNet Architecture

An effective neural network architecture for mobile and edge devices is called MobileNet. It is lightweight. It makes use of depthwise separable convolutions, which are made up of a 1x1 pointwise convolution after a depthwise convolution. Because of this separation, MobileNet performs far better computationally than regular convolutions, which makes it a good choice in situations when computational resources are limited. A significant reduction in the number of parameters is possible because to depthwise separable convolutions, which split the standard convolution into two independent operations. While the pointwise convolution aggregates the results across channels, the depthwise convolution works separately on

each input channel. By applying activation functions, this division not only minimizes the number of calculations but also creates non-linearity. As the foundational feature extractor for MobileNet SSD, MobileNet strikes a great mix between model size, accuracy, and inference performance.

SSD Head Architecture

Predicting bounding boxes and class scores at various scales throughout the feature maps produced by the MobileNet base is the responsibility of the SSD head. The fundamental principle of SSD is to enable the model to effectively capture items of diverse sizes by detecting objects at multiple scales in a single forward pass. In order to anticipate bounding boxes, the SSD head adds a number of convolutional layers with various aspect ratios. In order to ensure that the model can identify a wide variety of items inside an image, these layers are in charge of processing objects of various shapes. Each bounding box prediction is made up of four bounding box coordinates and a series of class scores that indicate how likely it is that the object will fall into each of the predetermined classes. SSD uses feature maps from various MobileNet architectural phases to capture data at many scales. The model will be able to identify both big and tiny things in an image thanks to this multi-scale technique.

5.1.2 Speed and Efficiency

The efficiency and speed of MobileNet SSD are among its main advantages. In comparison to conventional convolutional layers, the MobileNet base's usage of depthwise separable convolutions greatly lowers the computing cost. As a result, MobileNet SSD can operate in real time on mobile devices with modest processing power.

5.1.3 Object Detection

When it comes to object localization, MobileNet SSD excels, delivering precise bounding box predictions for items that are recognized. Bounding box predictions are made by the SSD architecture at numerous scales, which enables it to capture objects with different aspect ratios and sizes. This multi-scale method adds to the algorithm's resilience in identifying objects with various attributes. Numerous object categories can be handled by the algorithm. The model can recognize and categorize numerous items in a single image since the SSD head predicts class scores for each bounding box. Because of its adaptability, MobileNet SSD can be used for a wide range of applications, from general object detection to use cases specialized to the identification of vehicles or pedestrians.

5.1.4 Model Size

The comparatively small model size of MobileNet SSD is essential for deployment on devices with limited resources. Because it takes less memory and enables faster inference, the reduced model size makes it

perfect for real-time applications on mobile devices with constrained storage.

5.1.5 Why Choose MobileNet SSD

- Real-time Performance: Because MobileNet SSD is speed-optimized, it may be used in real-time applications like augmented reality, video analysis, and autonomous systems, where low latency is essential.
- Efficiency on Mobile Devices: Its architecture, which makes effective use of computational resources without compromising accuracy, is specifically designed for mobile and edge devices. This is especially evident in the usage of depthwise separable convolutions.
- Accurate Object Localization: MobileNet SSD's SSD design allows for accurate object localization at several scales, which makes it dependable for identifying objects in a range of situations.
- Versatility: The model can be applied to a multitude of use cases without requiring a significant amount of retraining because it can handle a broad range of object classifications.
- Compact Model Size: This model's compact size allows for great performance while making optimal use of resources, making it ideal for deployment on devices with constrained memory and storage.

5.2 YOLOV8

YOLO, which stands for "You Only Look Once," is a family of real-time object detection algorithms in computer vision. The YOLO algorithm is designed to detect and classify objects in images or video frames efficiently and accurately. The key idea behind YOLO is that it performs object detection in a single pass through the neural network, making it particularly well-suited for real-time applications.

5.2.1 Architecture

As a single-stage object detector, YOLO V8 comprises three key components: a backbone, a neck, and a head. The backbone, leveraging a pre-trained network, extracts high-level image features, employing residual and dense blocks to address issues like vanishing gradients. The model neck facilitates feature generalization to objects of varying sizes, while the head, the final stage, employs convolutional layers, batch normalization, and Sigmoid linear units to produce outcomes such as classes, and bounding box scores.

Anchor-Free Detection: One key feature of YOLOv8 is its adoption of anchor-free detection. Unlike previous YOLO versions that relied on predefined anchor boxes, YOLOv8 predicts the object center directly. This anchor-free approach reduces the number of box predictions and accelerates the Non-Maximum Suppression (NMS) post-processing step.

New Convolutions: The YOLOv8 architecture incorporates changes in the convolutional layers to enhance feature extraction. The stem's initial 6x6 convolution is replaced by a 3x3 convolution, and the C2f module replaces the C3 module. The C2f module (cross-stage partial bottleneck with two convolutions) plays a crucial role in combining high-level features with contextual information, contributing to improved detection accuracy.

Mosaic Augmentation: Training in YOLOv8 involves innovative techniques like mosaic augmentation. This process involves stitching together four images, providing the model with exposure to diverse scenarios, including variations in object locations, partial occlusion, and different surroundings. Mosaic augmentation aids in improving the model's ability to generalize to real-world conditions.

5.2.2 Speed and Efficiency

With a streamlined design and innovative techniques such as anchor-free detection, YOLOv8 achieves remarkable processing speeds, ensuring near-instantaneous predictions for object localization and classification. The model's efficiency is particularly notable in scenarios demanding rapid and accurate identification of objects, such as in autonomous driving or surveillance applications. The anchor-free approach reduces the computational burden by predicting the object center directly, leading to fewer box predictions and expediting post-processing steps. These advancements in speed and efficiency empower YOLOv8 to handle diverse tasks efficiently, solidifying its position as a leading solution in the field of computer vision and artificial intelligence.

5.2.3 Object Detection

YOLOv8 excels at object detection, a more sophisticated method that identifies and locates multiple objects within an image. This approach not only assigns class labels to objects but also furnishes precise bounding box coordinates, enabling accurate localization of each detected object. YOLOv8's proficiency in object detection makes it a robust solution for applications such as autonomous driving, robotics, and video surveillance.

5.2.4 Model Size

One notable feature of YOLOv8 is its ability to maintain high accuracy while keeping a compact model size. The YOLOv8 model size is optimized to strike a balance between computational efficiency and performance. This is crucial for real-time applications, where the model needs to be deployed on devices with limited computational resources.

The YOLOv8 architecture achieves this by implementing efficient design choices, such as utilizing a modified CSPDarknet53 backbone and incorporating anchor-free detection. The anchor-free model directly predicts the center of an object, eliminating the need for anchor boxes and reducing the number

of parameters in the model. This not only accelerates inference speed but also contributes to a more streamlined model size.

The model's compactness is further enhanced by the use of lightweight components like the C2f module, which efficiently combines high-level features with contextual information. Additionally, YOLOv8 leverages advanced techniques such as the Spatial Pyramid Pooling Fast (SPPF) layer, which accelerates computation by pooling features into a fixed-size map, optimizing the model for real-time processing.

The emphasis on model size in YOLOv8 is pivotal for deployment across various platforms, ensuring efficient utilization of resources without compromising detection accuracy. This characteristic makes YOLOv8 well-suited for applications in scenarios where computational resources are a critical consideration, such as edge devices and embedded systems.

5.2.5 Why Choose YOLOV8

- State-of-the-Art Accuracy: YOLOv8 stands out for its exceptional accuracy in object detection tasks, making it a top choice for applications demanding high precision.
- Real-Time Processing: With a processing speed of 280 FPS on an NVIDIA A100 and TensorRT, YOLOv8 excels in real-time scenarios, providing rapid and efficient object detection.
- Flexibility in Vision Tasks: YOLOv8 supports various vision tasks, including object detection, segmentation, pose estimation, tracking, and classification, offering a versatile solution for diverse applications.
- Anchor-Free Model: YOLOv8's anchor-free model eliminates the need for anchor boxes, streamlining the detection process and reducing computational complexity.
- Compact Model Size: Despite its high accuracy, YOLOv8 maintains a compact model size, making it suitable for deployment on devices with limited computational resources, such as edge devices and embedded systems.
- Efficient Design Choices: The architecture incorporates efficient design choices, such as the modified CSPDarknet53 backbone, C2f module, and Spatial Pyramid Pooling Fast (SPPF) layer, optimizing both speed and accuracy.
- Unified Framework: YOLOv8 integrates various object detection methods, including classification, object detection, and image segmentation, into a unified framework, eliminating the need for separate networks.
- Anchor Box Visualization Improvement: YOLOv8 improves upon the challenge of anchor boxes, providing better representation of the target distribution on custom datasets, enhancing the model's adaptability.

- Easy Integration and Deployment: YOLOv8 offers ease of use with a command line interface (CLI), PIP package installation, and multiple integrations for labeling, training, and deployment, enhancing its accessibility for developers and researchers.

Chapter 6

Methodology

6.1 Seminar Work (Research)

6.1.1 Super-Resolution Methodology

In addressing the challenge of real-world low-resolution images for PPE detection, this methodology integrates Generative Adversarial Networks (GANs) to enhance image quality. Due to resource constraints, a pre-existing dataset was utilized, acknowledging limitations in preparing and training an extensive dataset for the task.

Given resource limitations, we compared three GAN models for super-resolution: Progressive Growing GANs (PGGANs), Enhanced Super-Resolution GAN (ESRGAN), and a simplified GAN used for generating fake numbers. Recognizing the computational demands of training, particularly with the simple GAN, the choice considered the trade-off between model complexity and available resources.

Given resource limitations, we compared three GAN models for super-resolution: Progressive Growing GANs (PGGANs), Enhanced Super-Resolution GAN (ESRGAN), and a simplified GAN used for generating fake numbers. Recognizing the computational demands of training, particularly with the simple GAN, the choice considered the trade-off between model complexity and available resources.

In the initial stage of our methodology, we leverage a degradation GAN to synthesize realistic low-resolution images from high-resolution counterparts. This degradation GAN, by emulating real-world factors like motion blur and compression artifacts, enhances the authenticity of the low-resolution images. The goal is to create a challenging yet plausible dataset that closely mimics the conditions encountered in real-world scenarios.

Subsequently, these authentically degraded low-resolution images, alongside their corresponding high-resolution counterparts, are fed into the models of two prominent Generative Adversarial Networks (GANs): Enhanced Super-Resolution GAN (ESRGAN) and Progressive Growing GANs (PGGANs).

This comparative evaluation aims to discern which GAN model performs better in the task of super-resolving low-quality images.

ESRGAN is renowned for its capacity to generate high-quality and visually appealing super-resolved images. PGGANs, on the other hand, are celebrated for their progressive training approach, allowing for the generation of high-resolution images with intricate details. By subjecting the degraded low-resolution images to both ESRGAN and PGGAN, we seek to determine the superior model in terms of image fidelity, resolution, and overall visual quality.

This comparative analysis serves as a crucial step in our methodology, guiding the selection of the most effective GAN model for super-resolving low-quality images. The outcome of this evaluation will influence the subsequent stages of our approach, optimizing the quality of images fed into the computer vision model designed for PPE detection.

In the first phase of our methodology, the super-resolved images generated by the selected GAN model, be it ESRGAN or PGGAN, are seamlessly integrated into a dedicated computer vision model designed for PPE detection. Leveraging the improved image quality obtained through the GANs, our computer vision pipeline is equipped to more accurately and robustly identify and classify Personal Protective Equipment (PPE). The high-resolution images contribute to enhanced feature extraction, enabling the computer vision model to discern subtle details and patterns associated with different PPE types, thus improving the overall accuracy of the detection system.

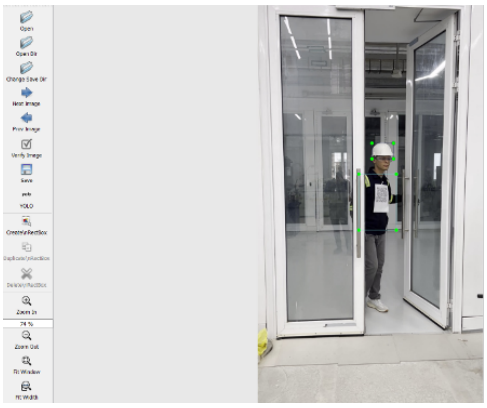
Looking ahead, the future plan involves scaling up our approach with larger resources. Despite the constraints faced in the current implementation, the success of the methodology highlights its potential for broader application. With increased resources, we aim to further refine our GAN models, potentially exploring more sophisticated architectures and training strategies to achieve even higher-quality super-resolved images. Additionally, a larger dataset can be employed for training, allowing for more diverse and representative scenarios. This scalability will not only improve the performance of the existing methodology but also open avenues for exploring more complex and demanding real-world applications in computer vision and image processing.

6.1.2 Computer Vision Methodology

We initiated our study by capturing a video featuring a subject wearing Personal Protective Equipment (PPE). This encompassed various safety gear such as helmets, gloves, masks, etc., essential for different industrial and healthcare applications. And a frame from the video we captured can be shown in figure 6.1a.

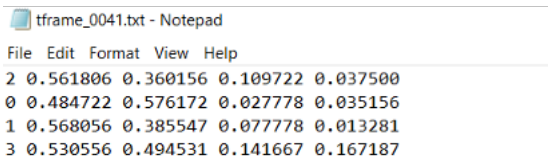
To facilitate the training of our computer vision models, we utilized the labeling application. This tool enabled us to annotate the images by drawing bounding boxes around the PPE items in the video frames. The chosen annotation format was YOLO (You Only Look Once), and the results were saved in text (txt) files. These files contained information about the location and class of each annotated object. As we can see in figure 6.1b is the application itself with an image we labelled. Also, in figure **??** is a result what the application does as it saves the bounding box in format of txt.

(a)



(b)

Figure 6.1



tframe_0041.txt - Notepad

File Edit Format View Help

```
2 0.561806 0.360156 0.109722 0.037500
0 0.484722 0.576172 0.027778 0.035156
1 0.568056 0.385547 0.077778 0.013281
3 0.530556 0.494531 0.141667 0.167187
```

Figure 6.2: YOLO format(.txt)

Our first model in the computer vision pipeline was YOLOv8, a state-of-the-art object detection algorithm known for its accuracy and efficiency. Using the annotated dataset in YOLO format, the model underwent training. Over multiple iterations, YOLOv8 learned to recognize and localize PPE items in the images, refining its performance as we will discuss this in the discussion and results.

To diversify our approach, we incorporated MobileNet SSD as the second model in our study. However, MobileNet SSD required annotations in XML format. To address this, we developed a custom script to convert the existing YOLO format annotations (in txt files) into XML format. This conversion step facilitated a seamless transition from YOLOv8 to MobileNet SSD. The code used to make this change is shown in (Figure XX). As we can see we used Pylabel, which is a library in python that turn the .txt files to XML format to be compatible with MobileNet SSD. Following the annotation of images in XML format for MobileNet SSD, we converted the annotated data into TFRecord format for TensorFlow compatibility during training. A custom script parsed XML files, extracting relevant information like bounding box coordinates and class labels. This data was then encoded into TFRecord, a binary format that enhances storage efficiency and supports parallelization for faster loading during training.

```
for path in path_to_annotations:
    dataset = importer.ImportYoloV5(path=path, path_to_images=path_to_images, cat_names=yoloclasses,
    img_ext="jpg", name="coco128")
    print(f"Number of images: {dataset.analyze.num_images}")
    print(f"Number of classes: {dataset.analyze.num_classes}")
    print(f"Classes:{dataset.analyze.classes}")
    print(f"Class counts:\n{dataset.analyze.class_counts}")
    output_path = str(Path(dataset.path_to_annotations, "../", "voc_annotations/"))
    Path(output_path).mkdir(exist_ok=True)
    dataset.export.ExportToVoc(output_path=output_path)[0]

Importing YOLO files...: 100%|                                                    | 82/82 [00:00<00:00, 84.89it/s]


Number of images: 82
Number of classes: 4
Classes:['gloves', 'Googles', 'Helmets', 'Vests']
Class counts:
cat_name
gloves     146
Helmets     81
Vests       81
Googles     75
Name: count, dtype: int64


Exporting VOC files...: 100%|                                                     | 82/82 [00:00<00:00, 143.36it/s]
```

Figure 6.3: Turning from txt to xml format

With the dataset now compatible with MobileNet SSD, we initiated the training process for our second model. MobileNet SSD, known for its speed and efficiency, was used to identify and localize PPE items based on the annotated dataset. Through iterative training, the model enhanced its ability for object detection.

6.2 PBL Work (Implementation)

6.2.1 Overview

Introduction

Our project consists of two main components: the AI part and the mobile application part. These components are interconnected via Google's Firebase, facilitating seamless integration and data flow between them. After thorough research and consideration of various methodologies, we chose not to use Generative Adversarial Networks (GANs) due to several limitations. Instead, we opted for interpolation techniques, which, while not as accurate as GANs, proved sufficient for our needs.

AI Component

The AI component centers around a computer vision model designed to detect and annotate objects in real-time. We leveraged a YOLO (You Only Look Once) detection model due to its efficiency and accuracy in object detection tasks. Our algorithm is meticulously crafted to identify various safety equipment, such as helmets, gloves, vests, and goggles, in an industrial setting. The model's outputs are processed to ensure precise detection and annotation, which are critical for monitoring safety compliance.

Mobile Application

The mobile application serves as the user interface, allowing users to view the annotated images and receive notifications about any detected safety violations. The app is designed to be user-friendly, providing real-time updates and insights into the monitored environment. This interface is crucial for supervisors and safety officers to ensure compliance with safety protocols and to address any issues promptly.

Integration via Firebase

Firebase acts as the bridge between the AI component and the mobile application. It handles data storage, real-time synchronization, and notification services. The detected data from the AI model is uploaded to Firebase, which then pushes the information to the mobile application. This integration ensures that users receive real-time updates on safety compliance, making the system robust and efficient.

Real-Time Streaming

An essential aspect of our system is real-time streaming, which ensures that video data from the monitored environment is continuously fed into the AI model for processing. This streaming capability allows for live monitoring and instant detection of safety compliance issues. The video feed is captured, processed by the AI model for detection and annotation, and then streamed to the mobile application via

the streaming server. This real-time feedback loop is crucial for maintaining up-to-date safety monitoring and provides immediate visibility into potential safety violations. By leveraging efficient streaming protocols, we ensure minimal latency and high-quality video transmission, making the system highly responsive and effective.

Methodological Decision: Interpolation vs. GANs

Initially, we considered using GANs to enhance our detection model. However, after extensive research, we identified several challenges:

- Data Specificity: GANs require a large and specific dataset, which involves collecting high-quality and low-quality images from various factory perspectives.
- Resource Intensity: Training GANs is computationally expensive and resource-intensive, which is beyond our current capabilities.
- Feasibility: The practical constraints of acquiring the necessary data and computational resources made the use of GANs unfeasible.

Given these challenges, we decided to use interpolation techniques. Although less accurate than GANs, interpolation provided a practical solution that met our requirements. This method allowed us to enhance image quality to a satisfactory level without the significant overhead associated with GANs.

6.2.2 FireBase

This section of the report explains how the provided Python code links to the Firestore database and uploads detection data from a YOLOv8 model. The data is then accessible by a mobile application, which presents it to users. The integration of the Firestore is crucial for the seamless storage, retrieval, and management of data in real-time.

Libraries

1. OS:
   - **Module**: `import os`
   - **Role**: Provides functionalities to interact with the operating system. It is used for file path manipulations and other OS-level operations.
   - **Functions Used**: `os.path.join` for constructing file paths.
2. Datetime :
   - **Module**: `from datetime import datetime`

- **Role**: The datetime module supplies classes for manipulating dates and times. It is used to timestamp the detection data before uploading it to Firestore.
- **Functions Used**: `datetime.now()` to get the current date and time.

3. Firebase Admin SDK:

- **Modules**: `import firebase`$_{a}dminand$

Process

1. **Firebase Admin SDK Setup**: The code begins by importing necessary libraries and initializing the Firestore database using Firebase Admin SDK. The Firebase Admin SDK allows for secure server-side access to Firestore, a NoSQL document database. Here in the figure below (figure 6.4) , the credentials.Certificate function loads the service account key file, which contains the credentials necessary to authenticate and authorize access to the Firestore database. `The firebase_admin.initialize_app(cred)` function initializes the app with these credentials, and `firestore.client()` returns a Firestore client for interacting with the database.

```
# Initialize Firestore
cred = credentials.Certificate('officialpbl-firebase-adminsdk-3qtj6-7e0d0b4e07.json')
firebase_admin.initialize_app(cred)
db = firestore.client()
```

Figure 6.4: Credentials Access

2. **Data Upload Function**: A function named `update_or_create_document` ,shown in figure 6.5, is defined to update or create a document within a specified Firestore collection. This function takes three parameters:

   - collection_name: The name of the Firestore collection.
   - document_id: The ID of the document within the collection.
   - data: The data to be stored in the document. The set method is used with the merge=True option to ensure that existing documents are updated with the new data without overwriting other fields.

```python
def update_or_create_document(collection_name, document_id, data):
    """
    Updates an existing document or creates a new document in a Firestore collection.

    This function sets the data for a specified document in a given collection.
    If the document does not exist, it will be created. If the document already
    exists, the provided data will be merged with the existing data.

    Parameters:
    collection_name (str): The name of the Firestore collection.
    document_id (str): The ID of the document to update or create.
    data (dict): The data to set in the document. If merging with an existing document,
                 this data will be combined with the existing data.

    Returns:
    None
    """
    # Reference the document in the specified collection
    doc_ref = db.collection(collection_name).document(document_id)

    # Set the data for the document, merging with existing data if the document exists
    doc_ref.set(data, merge=True)
```

Figure 6.5: Code

3. `upload_Image_FireBase` : The `upload_Image_FireBase` function is designed to upload an image to a Firestore collection, storing it along with metadata such as the error message and timestamp. Here's a detailed breakdown of how it works:

   - `collection_name` (str): The name of the Firestore collection where the image will be stored.
   - `document_id` (str): The ID of the document to be created or updated within the collection.
   - `cropped_img` (numpy.ndarray): The cropped image array that needs to be uploaded.
   - `error_message` (str): An error message to be associated with the image in the Firestore document.

   and then we can talk about the process :

   (a) **Flatten Image**: The cropped image is flattened into a one-dimensional list to make it compatible with Firestore's data storage format.

   (b) **Timestamp**: The current timestamp is obtained using `datetime.now()` to record when the image was uploaded.

   (c) **Document Reference**: A reference to the specific Firestore document is created using the collection name and document ID.

   (d) **Set Document Data**: The document is populated with the flattened image data, the image's shape (for later reconstruction), the error message, and the timestamp.

   (e) **Upload to Firestore**: The document is uploaded to Firestore using the `set` method on the document reference.

   This function ensures that each image is stored efficiently and is accompanied by relevant metadata for later retrieval and analysis.

```python
def upload_Image_FireBase(collection_name, document_id, cropped_img, error_message):
    """
    Uploads a image to Firestore.

    Args:
        collection_name (str): Name of the Firestore collection.
        document_id (str): ID of the document to be created/updated.
        cropped_img (numpy.ndarray): Cropped image array.
        error_message (str): Error message to be stored in the document.

    Returns:
        None
    """
    # Flatten the cropped image and convert it to a list
    cropped_img_list = cropped_img.flatten().tolist()

    # Get the current timestamp
    current_time = datetime.now()

    # Construct the document reference
    doc_ref = db.collection(collection_name).document(document_id)

    # Set document data
    doc_ref.set({
        'image_url': cropped_img_list,
        'shape': cropped_img.shape,
        'error': error_message,
        'data_time': current_time
    })
```

Figure 6.6: Code of `upload_Image_FireBase`

4. `retrieve_and_save_image`: The `retrieve_and_save_image` function is responsible for retrieving an image from a Firestore collection, reshaping it to its original dimensions, and saving it as a file. Here's a detailed explanation of its functionality:

   - `collection_name` (str): The name of the Firestore collection from which the image will be retrieved.
   - `document_id` (str): The ID of the document containing the image.
   - `save_path` (str): The file path where the retrieved image will be saved.

   and here is the process of the function:

   (a) **Access Firestore**: A Firestore client is initialized to interact with the Firestore database.

   (b) **Document Reference**: A reference to the specific document within the specified collection is created using the collection name and document ID.

   (c) **Retrieve Document Data**: The data from the document is fetched and converted to a dictionary format.

   (d) **Reshape Image**: The flattened image array retrieved from Firestore is reshaped back to its original dimensions using the stored shape information.

   (e) **Save Image**: The reshaped image is saved to the specified file path using OpenCV's `cv2.imwrite` function.

   (f) **Delete Document**: The document is deleted from Firestore to manage storage and ensure that the same image is not redundantly stored.

   This function allows efficient retrieval and reconstruction of images from Firestore, ensuring that the data is stored compactly and can be easily accessed and utilized when needed.

```python
def retrieve_and_save_image(collection_name, document_id, save_path):
    """
    Retrieves an image from Firestore, reshapes it, and saves it as a file.

    Args:
        collection_name (str): Name of the Firestore collection.
        document_id (str): ID of the document containing the image.
        save_path (str): Path to save the image file.

    Returns:
        None
    """
    # Access Firestore
    db = firestore.client()

    # Access the specified collection
    collection_ref = db.collection(collection_name)

    # Access the document within the collection
    doc_ref = collection_ref.document(document_id)

    # Retrieve the data from the document
    doc_data = doc_ref.get().to_dict()

    # Get the original shape of the image
    original_shape = doc_data['shape']

    # Retrieve the flattened image array and reshape it
    original_array = np.array(doc_data['image_url']).reshape(original_shape)

    # Save the image
    cv2.imwrite(save_path, original_array)

    # Delete the document from Firestore
    doc_ref.delete()
```

Figure 6.7: code of `retrieve_and_save_image`

5. **Sending Data to Firestore**: After processing each frame, the detection data is prepared and sent to Firestore. and we have 2 types of the data.

   - First, We send the number of the objects and humans in firebase and that happens as A dictionary named detection_data is created, containing the timestamp and counts of detected objects. This data is then uploaded to the Firestore collection 'notifications_' under the document ID 'detection_data' using the previously defined function.

```python
# Send detected data to Firestore
detection_data = {
    'timestamp': datetime.now(),
    'num_persons': len(humans),
    'num_gloves' : len(gloves),
    'num_googles': len(googles),
    'num_helmets': len(helmets),
    'num_vests': len(vest)
}

# Update or create a document in Firestore
update_or_create_document('notifications_', 'detection_data', detection_data)
```

Figure 6.8: number of objects and humans being sent to Firebase

   - Second , we send the errors we found in database. a dictionary named errors_data is created, and it contains the timestamp and image, and image_shape, so the mobile application can show to us the image when an error happened with a message what is the error.

Importance of Firestore Database

1. Scalability: Firestore is designed to scale seamlessly, accommodating the growing data needs without compromising performance. This is crucial for applications that may experience variable loads and need to handle large amounts of data efficiently.

2. Real-time Synchronization: One of the key features of Firestore is its ability to synchronize data in real-time across multiple devices. This ensures that any updates to the detection data are instantly reflected on the mobile application, providing users with up-to-date information.

3. Flexible Data Structure: Firestore's NoSQL document-based structure allows for flexible and dynamic data models. This flexibility is essential for handling diverse data types and structures, such as detection data from different objects.

4. Ease of Integration: The Firestore client libraries and SDKs simplify the integration process, allowing developers to focus on the application logic rather than the underlying infrastructure. The Python Firebase Admin SDK provides robust functionalities to interact with the Firestore database securely and efficiently.

5. Security: Firestore provides robust security features, including granular access control with Firebase Authentication and Firestore Security Rules. This ensures that data is protected against unauthorized access and modifications.

6.2.3 Streaming

Overview

This section of the report involves a Detailed explanation to the comprehensive streaming setup that uses MJPEG streaming technology for real-time video transmission. where a Raspberry Pi is connected to a USB camera (can be replaced with Ip camera) and streams it to the processing unit. The Processing unit processes the video frames using the YOLOv8 object detection model and streams the annotated frames to a mobile application.

Detailed Description

The project is divided into several components:

1. **Raspberry Pi Camera Streaming**

   The Raspberry Pi captures video from a connected camera and streams it to a laptop using MJPEG streaming. The MJPEG stream transmits each frame as a JPEG image over HTTP. The setup involves creating a streaming server that listens for incoming connections and continuously captures and encodes frames from the camera. These frames are then sent over HTTP to clients that connect to the streaming server. This allows for real-time viewing of the camera feed on the laptop.

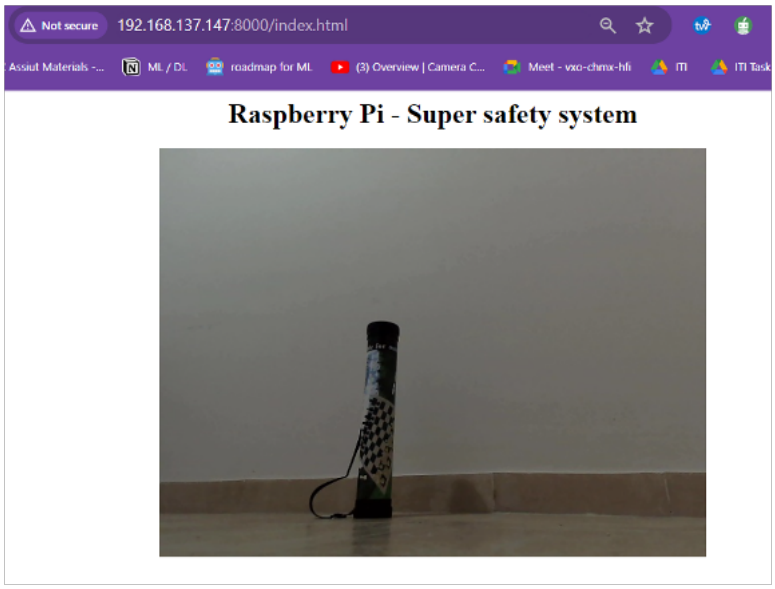


Figure 6.9: Raspberry Pi stream

2. **Processing unit (Laptop): Processing and Streaming**

   The laptop receives the MJPEG stream from the Raspberry Pi, processes each frame using the YOLOv8 model, and streams the results using the mjpeg-streamer Python module to a mobile application.

3. **Set Up Stream**

   To initialize the streaming server on the laptop. It creates a Stream object to handle the video stream, sets up the MJPEG server with the specified IP address and port, and starts the server to begin streaming and returns the streamer object

```
# Initialize the camera stream with specified size, quality, and frame
    rate
streamer = Stream("my_camera", size=(w, h), quality=40, fps=15)
# Set up the MJPEG server with the given IP and port
server = MjpegServer(ip, port)
# Add the camera stream to the server
server.add_stream(streamer)
# Start the server to begin streaming
server.start()
# Return the Stream object
return streamer
```

4. **Stream Frame**

   Updates the current frame for the specified streamer to broadcast. It ensures that the latest frame is set for streaming, allowing real-time updates to be viewed by clients:

```
def stream_frame(streamer, frame):
# Set the current frame for the streamer
streamer.set_frame(frame)
```

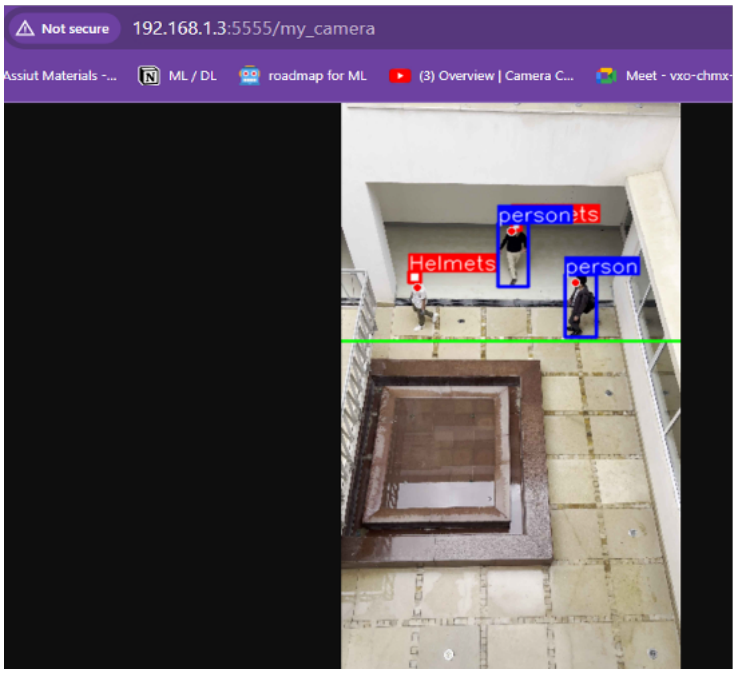


Figure 6.10: Stream frame

Libraries Used

1. **OpenCV**: For capturing video frames from the MJPEG stream and processing the images.
2. **YOLOv8**: For object detection on the video frames.
3. **mjpeg-streamer**: For setting up the MJPEG stream from the processing unit (laptop) to the mobile application.
4. **SocketServer and HTTPServer**: For handling HTTP requests and serving the MJPEG stream from the Raspberry Pi.

Conclusion

This chapter showcases our end-to-end solution for real-time video streaming and processing using MJPEG technology. The Raspberry Pi captures and streams video to a laptop, which processes the frames using the YOLOv8 object detection model and streams the annotated frames to a mobile application. This setup highlights the integration of hardware and software components to achieve efficient and effective real-time video analysis and streaming.

6.2.4 YOLO and AI

The AI component of our project is centered around leveraging computer vision techniques to detect and annotate objects within images, with a specific focus on safety equipment and body positions. The primary tasks include object detection using a YOLO model, image preprocessing to ensure optimal input for the model, and post-detection analysis to verify the presence and position of critical body parts. Here is All of the AI part:

Libraries

The implementation relies on several Python libraries to perform various tasks, including computer vision, pose estimation, object detection, and visualization. Below is an overview of the libraries used:

- OpenCV (cv2) OpenCV, or Open Source Computer Vision Library, is a popular open-source computer vision and machine learning software library. It provides a wide range of functionalities for image and video analysis, including image processing, feature detection, object recognition, and more. In this project, OpenCV is utilized for tasks such as image loading, manipulation, and drawing on images.

- Math (math) The math library is a built-in Python module that provides mathematical functions and constants for numerical computations. It is used in the project for basic mathematical operations and calculations, such as computing distances between landmarks or performing threshold comparisons.

- (mediapipe) MediaPipe is an open-source framework developed by Google that offers various pre-built solutions for media processing tasks, including real-time pose estimation, hand tracking, face detection, and more. In this project, the MediaPipe library is employed for pose estimation, which involves detecting and tracking key body landmarks from images or video streams.

- Ultralytics YOLO (YOLO) Ultralytics YOLO is an implementation of the You Only Look Once (YOLO) object detection algorithm developed by Ultralytics. YOLO is a state-of-the-art real-time object detection system that can detect multiple objects in images or video frames with high accuracy and efficiency. The Ultralytics YOLO library is utilized in the project for detecting objects such as persons and helmets within the images.

- Ultralytics Annotator The Ultralytics Annotator is a component of the Ultralytics YOLO library used for visualizing and annotating detected objects in images or video frames. It provides functionalities for drawing bounding boxes, labels, and other annotations on images, making it easier to interpret and analyze the detection results.

Functions Explaination

1. **Detection Model**:

   (a) *DetectionModel Class*: The `DetectionModel` class encapsulates a YOLO detection model. It is designed to load a YOLO model and facilitate predictions on input images.

   - Initialization: The `DetectionModel` class is initialized with the following parameters:
     - `model_path (str)`: The path to the YOLO model.
     - `classes (list, optional)`: A list of class names for the model. If provided, it sets the classes attribute of the YOLO model.

     Upon initialization, the YOLO model is loaded from the specified path, and the classes attribute is set if provided.

```
class DetectionModel:
    """
    Class for YOLO detection models.
    """
    def __init__(self, model_path, classes=None):
        """
        Initializes the DetectionModel object.

        Args:
            model_path (str): Path to the YOLO model.
            classes (list): List of classes for the model.
        """
        self.model = YOLO(model_path)
        if classes:
            self.model.classes = classes
```

Figure 6.11: Init in Class

   - *predict_and_annotate Method* The `predict_and_annotate` method performs object detection on an input image and annotates the detected objects.
   - Arguments
     - `img (numpy.ndarray)`: The input image on which predictions are to be made.
     - `conf_threshold (float)`: The confidence threshold for the detection. Only detections with a confidence score higher than this threshold will be considered.
     - `color (tuple, optional)`: The color of the bounding boxes for the annotations. The default is red (0, 0, 255).
   - Process
     i. The method calls the `predict` method to obtain detection results.
     ii. An `Annotator` object is initialized with the input image.
     iii. The method iterates over the detection results, extracting bounding boxes and class indices.
     iv. If a detected object's confidence score exceeds the specified threshold, a bounding box and label are added to the `Annotator` object.

```python
def predict_and_annotate(self, img, conf_threshold, color=(0, 0, 255)):
    """
    Predicts objects in the image and annotates them.

    Args:
        img (numpy.ndarray): Input image.
        conf_threshold (float): Confidence threshold for detection.
        color (tuple): Bounding box color.

    Returns:
        numpy.ndarray: Bounding box coordinates.
        Annotator: Annotator object with annotated image.
    """
    results = self.predict(img)
    annotator = Annotator(img)
    for result in results:
        boxes = result.boxes
        for box in result.boxes:
            if box.conf > conf_threshold:
                box_coords = box.xyxy[0]
                class_index = box.cls
                annotator.box_label(box_coords, self.model.names[int(class_index)], color=color)
    return boxes, annotator
```

Figure 6.12: Predict and annotate function in Class

```python
def predict(self, img):
    """
    Performs prediction on the image.

    Args:
        img (numpy.ndarray): Input image.

    Returns:
        List: Prediction results.
    """
    return self.model.predict(img)
```

Figure 6.13: Perdict Function in Class

- Returns
  - `numpy.ndarray`: The bounding box coordinates of the detected objects.
  - `Annotator`: An `Annotator` object with the annotated image.

(b) *predict Method*: The `predict` method performs prediction on an input image.

- Arguments
  - `img (numpy.ndarray)`: The input image on which predictions are to be made.
- Process The method uses the YOLO model to predict objects in the input image.
- Returns
  - `List`: A list containing the prediction results.

2. **Preprocessing Frame**: The `preprocess_frame` function preprocesses the input frame by optionally flipping it horizontally and/or vertically and resizing it.

   - Arguments
     - `frame (numpy.ndarray)`: The input frame.
     - `flip_horizontal (bool, optional)`: Whether to flip the frame horizontally. Default is True.
     - `flip_vertical (bool, optional)`: Whether to flip the frame vertically. Default is True.

```python
def preprocess_frame(frame, flip_horizontal=True, flip_vertical=True, resize_scale=0.15):
    """
    Preprocesses the input frame.

    Args:
        frame (numpy.ndarray): The input frame.
        flip_horizontal (bool): Whether to flip the frame horizontally. Default is True.
        flip_vertical (bool): Whether to flip the frame vertically. Default is True.
        resize_scale (float): Scaling factor for resizing the frame. Default is 0.15.

    Returns:
        numpy.ndarray: Preprocessed image in RGB format.
        numpy.ndarray: Preprocessed frame.
    """
    # Flip the frame horizontally if specified
    if flip_horizontal:
        frame = cv2.flip(frame, X_MIN_INDEX)

    # Flip the frame vertically if specified
    if flip_vertical:
        frame = cv2.flip(frame, Y_MIN_INDEX)

    # Resize the frame
    frame = cv2.resize(frame, (0, 0), fx=resize_scale, fy=resize_scale)

    # Convert the frame to RGB format
    img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    return img, frame
```

Figure 6.14: Preprocess Function

  - `resize_scale (float, optional)`: The scaling factor for resizing the frame. Default is 0.15.

- Process
  (a) The frame is flipped horizontally if specified.
  (b) The frame is flipped vertically if specified.
  (c) The frame is resized according to the specified scaling factor.
  (d) The frame is converted from BGR to RGB format.
- Returns
  - `numpy.ndarray`: The preprocessed image in RGB format.
  - `numpy.ndarray`: The preprocessed frame.

3. **Checking Body Part Presence**: The `check_body_part_presence` function checks if a specified body part (e.g., helmet) is present near a person's body landmark.

   - Arguments
     - `result (object)`: The result object containing pose landmarks.
     - `body_index (int)`: The index of the body landmark in the result object.
     - `box_coords (list)`: The bounding box coordinates of the person.
     - `body_parts (list)`: A list of bounding boxes for the body parts to check (e.g., helmets).
     - `img_shape (tuple)`: The shape of the image.
     - `DISTANCE_THRESHOLD (float)`: The distance threshold to consider a body part near the body landmark.
   - Process
     (a) The method retrieves the coordinates of the specified body landmark.

```python
def check_body_part_presence(result, body_index, box_coords, body_parts, img_shape, DISTANCE_THRESHOLD):
    """
    Checks if a specified body part (e.g., helmet) is present near a person's body landmark.

    Args:
        result (object): Result object containing pose landmarks.
        body_index (int): Index of the body landmark in the result object.
        box_coords (list): Bounding box coordinates of the person.
        body_parts (list): List of bounding boxes for the body parts to check (e.g., helmets).
        img_shape (tuple): Shape of the image.
        DISTANCE_THRESHOLD (float): The distance threshold to consider a body part near the body landmark.

    Returns:
        bool: True if a body part is present, False otherwise.
    """
    # Get the landmark of the person's body part using the body index
    body_landmark = result.pose_landmarks.landmark[body_index]

    # Extract image height, width, and number of channels from img_shape
    ih, iw, _ = img_shape

    # Calculate the x and y coordinates of the body landmark in the image
    body_x, body_y = int(body_landmark.x * iw), int(body_landmark.y * ih)

    # Add the x and y coordinates of the bounding box to get the absolute position of the body landmark
    body_x += int(box_coords[X_MIN_INDEX])
    body_y += int(box_coords[Y_MIN_INDEX])

    # Iterate through each body part bounding box
    for part_coord in body_parts:
        # Calculate the center coordinates of the body part
        part_center_x = (part_coord[X_MIN_INDEX] + part_coord[X_MAX_INDEX]) / 2
        part_center_y = (part_coord[Y_MIN_INDEX] + part_coord[Y_MAX_INDEX]) / 2

        # Calculate the distance between the body landmark and the center of the body part
        distance = math.sqrt((body_x - part_center_x)**2 + (body_y - part_center_y)**2)

        # Check if the distance is less than the threshold or if the body landmark is within the body part bounding box
        if (distance < DISTANCE_THRESHOLD or
                (part_coord[X_MIN_INDEX] < body_x < part_coord[X_MAX_INDEX] and
                 part_coord[Y_MIN_INDEX] < body_y < part_coord[Y_MAX_INDEX])):
            return True  # Body part is present
```

Figure 6.15: Checking body parts function

(b) It iterates through each body part's bounding box, calculating the distance between the body landmark and the center of the body part.

(c) If the distance is less than the threshold or if the body landmark is within the body part bounding box, it returns True indicating the presence of the body part.

- Returns
  - `bool`: True if the body part is present, False otherwise.

4. **Checking Leg Position**: The `check_leg_position` function checks if a person's leg position is above a certain threshold.

   - Arguments
     - `leg_landmark (Landmark)`: The landmark of the person's leg.
     - `box_coords (list)`: The bounding box coordinates of the person.
     - `img_shape (tuple)`: The shape of the image.
     - `frame (numpy.ndarray)`: The input frame.
     - `not_passing (bool)`: A flag indicating if the person is not passing the threshold.
   - Process

     (a) The method calculates the x and y coordinates of the leg in the image.

     (b) It adds the bounding box coordinates to get the absolute position of the leg.

     (c) If the leg's x-coordinate is above a certain threshold and the `not_passing` flag is False, a line is drawn on the frame to indicate the threshold.

     (d) The `not_passing` flag is updated to True.
   - Returns
     - `bool`: The updated `not_passing` flag.

```python
def check_leg_position(leg_landmark, box_coords, img_shape, frame, not_passing):
    """
    Checks if a person's leg position is above a certain threshold.

    Args:
        leg_landmark (Landmark): Landmark of the person's leg.
        box_coords (list): Bounding box coordinates of the person.
        img_shape (tuple): Shape of the image.
        frame (numpy.ndarray): Input frame.
        not_passing (bool): Flag indicating if the person is not passing the threshold.

    Returns:
        bool: Updated flag indicating if the person is not passing the threshold.
    """
    # Extract image height, width, and number of channels from img_shape
    ih, iw, _ = img_shape

    # Calculate the x and y coordinates of the leg in the image
    leg_x, leg_y = int(leg_landmark.x * iw), int(leg_landmark.y * ih)

    # Add the x and y coordinates of the bounding box to get the absolute position of the leg
    leg_x += int(box_coords[X_MIN_INDEX])
    leg_y += int(box_coords[Y_MIN_INDEX])

    # Check if the leg's x-coordinate is above a certain threshold
    if not_passing == False and leg_x > HALF_FRAME_HEIGHT:
        # Draw a line on the frame to indicate the threshold line
        cv2.line(frame, (0, HALF_FRAME_HEIGHT), (frame.shape[1], HALF_FRAME_HEIGHT), RED_COLOR, LINE_THICKNESS)
        # Update the not_passing flag to True since the person's leg is above the threshold
        not_passing = True

    # Return the updated not_passing flag
    return not_passing
```

Figure 6.16: Checking Leg Position Function

Algorithm for check_leg_position

1. Purpose: The check_leg_position function checks if a person's leg is positioned above a certain threshold, indicating whether they are in a restricted or unsafe area.

2. Steps:

   (a) Extract Image Dimensions: Retrieve the height (ih), width (iw), and channels from the image shape.

   (b) Calculate Leg Coordinates: Determine the x and y coordinates of the leg landmark using MediaPipe in the image by scaling the normalized coordinates with the image dimensions.

   (c) Check Leg Position: Compare the leg's x-coordinate against a predefined threshold which to know if it's passed the line or not. If the leg is above the threshold, mark the frame with a line and update the not_passing flag to True which means it passed the line.

3. Drawing: Here to explain more using drawings, in our first case is the worker is in the factory and he didn't pass the line that will make him unsafe , So, the line in our program is green as can see in the drawing and that's what we do in the code.

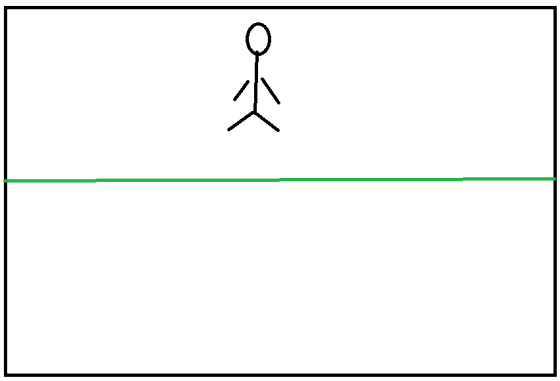


Figure 6.17: Case 1 - Design Part

and in the second case, as we see , if the leg part which is determined by blue dot passed the line , the line turns to red and send it to the firebase and it's an error that occurs.

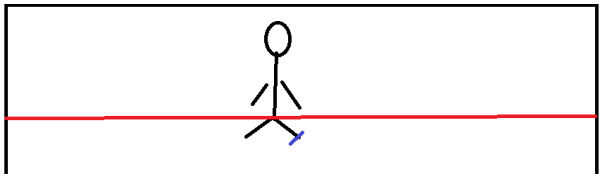


Figure 6.18: Case 2 - Design Part

Algorithm for check_body_part_presence

1. Purpose: The check_body_part_presence function determines if a specified body part (e.g., a helmet) is present near a person's body landmark by checking the proximity between detected body parts and landmarks.

2. Steps:

   - Get Landmark Coordinates: Retrieve the coordinates of the specified body landmark.
   - Extract Image Dimensions: Retrieve the height (ih), width (iw), and channels from the image shape.
   - Calculate Landmark Position: Determine the x and y coordinates of the landmark by scaling with image dimensions and adjusting with bounding box coordinates.
   - Iterate Over Body Parts: Loop through each detected body part's bounding box.
   - Calculate Distance: Compute the Euclidean distance between the body landmark and the center of the body part.
   - Check Proximity: Determine if the distance is less than the threshold or if the body landmark lies within the body part's bounding box. If true, return True indicating the presence of the body part.
   - Return Result: If no body part is within the threshold distance, return False.

3. Drawing: the case that will return true can seen below , as it's get the point of the nose from media pipe and then we try to detect helmets. so the point of the nose surely will be inside the helmet or beside it, so we use this case to try to know if he wear the helmet or not and we use it like this because if we use if it inside the body, because he may not wear the helmet but puts it in with hands.

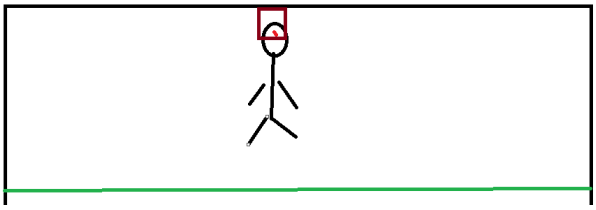


Figure 6.19: Helmet Case

6.2.5 Vision Transformers

Introduction

Object detection is a crucial task in computer vision, involving both the identification and localization of objects within images. Two notable methods for object detection are YOLO (You Only Look Once) and DETR (DEtection TRansformer). This section will discuss mainly the implemented Detr and provides a comprehensive comparison between these two approaches based on their architectures, methodologies, performance, and applications.

Overview and Explanation

This section outlines the process of setting up, training, and evaluating a DETR (Detection Transformer) model for object detection using PyTorch Lightning. The process includes:

1. **Setup and Downloading Data:**
   - Import necessary libraries.
   - Set the Roboflow API key securely.
   - Connect to the Roboflow workspace and project.
   - Download the dataset in COCO format.
2. **Creating DataLoaders:**
   - Define paths for training, validation, and test directories.
   - Create a custom ‘CocoDetection‘ class that extends ‘torchvision.datasets.CocoDetection‘.
   - Instantiate the training, validation, and test datasets using the custom ‘CocoDetection‘ class.
3. **Visualization:**
   - Randomly select an image from the training dataset.
   - Visualize the image with ground truth annotations using OpenCV and a custom ‘BoxAnnotator‘.

4. **Creating DataLoaders:**
   - Define a collate function to handle batching of images and annotations.
   - Create data loaders for training, validation, and test datasets.
5. **Training the Model with PyTorch Lightning:**
   - Define a ‘Detr‘ class that extends ‘pl.LightningModule‘.
   - Implement methods for the forward pass, training, and validation steps.
   - Configure the optimizer.

Setup and Downloading Data

1. **Importing Required Libraries and Setting API Key**

```
from getpass import getpass

ROBOFLOW_API_KEY = getpass('Enter ROBOFLOW_API_KEY secret value: ')
```

Listing 6.1: Setting the API key using `getpass`

2. **Connecting to Roboflow and Downloading Data**

```
from roboflow import Roboflow
rf = Roboflow(api_key=ROBOFLOW_API_KEY)
project = rf.workspace("cse-pbl").project("cse-pbl")
dataset = project.version(1).download("coco")
print(dataset.location)
dataset_path = dataset.location
```

Listing 6.2: Downloading the dataset from Roboflow

Creating DataLoaders

1. **Setting Up Directories and CocoDetection Class**

```
import os
import torchvision

ANNOTATION_FILE_NAME = "_annotations.coco.json"
TRAIN_DIRECTORY = os.path.join(dataset.location, "train")
VAL_DIRECTORY = os.path.join(dataset.location, "valid")
TEST_DIRECTORY = os.path.join(dataset.location, "test")
```

Listing 6.3: Setting up dataset directories

```
class CocoDetection(torchvision.datasets.CocoDetection):
    def __init__(self, image_directory_path: str, image_processor, train: bool = True):
      ....
    def __getitem__(self, idx):
        images, annotations = super(CocoDetection, self).__getitem__(idx)
        image_id = self.ids[idx]
        annotations = {'image_id': image_id, 'annotations': annotations}
        encoding = self.image_processor(images=images, annotations=annotations, return_tensors="pt")
        pixel_values = encoding["pixel_values"].squeeze()
        target = encoding["labels"][0]

        return pixel_values, target
```

Listing 6.4: Defining the CocoDetection class

2. **Creating Dataset Instances**

```
TRAIN_DATASET = CocoDetection(image_directory_path=TRAIN_DIRECTORY, image_processor=image_processor, train=True)
VAL_DATASET = CocoDetection(image_directory_path=VAL_DIRECTORY, image_processor=image_processor, train=False)
TEST_DATASET = CocoDetection(image_directory_path=TEST_DIRECTORY, image_processor=image_processor, train=False)
```

Listing 6.5: Creating train, validation, and test datasets

Visualization

Visualizing a Random Image

```
import random
import cv2
import numpy as np

image_ids = TRAIN_DATASET.coco.getImgIds()
image_id = random.choice(image_ids)
image = TRAIN_DATASET.coco.loadImgs(image_id)[0]
annotations = TRAIN_DATASET.coco.imgToAnns[image_id]
image_path = os.path.join(TRAIN_DATASET.root, image['file_name'])
image = cv2.imread(image_path)

detections = sv.Detections.from_coco_annotations(coco_annotation=annotations)
categories = TRAIN_DATASET.coco.cats
```

```
id2label = {k: v['name'] for k,v in categories.items()}
labels = [f"{id2label[class_id]}" for _, _, class_id, _ in detections]
box_annotator = sv.BoxAnnotator()
frame = box_annotator.annotate(scene=image, detections=detections, labels=labels)
```

Listing 6.6: Visualizing a random image with annotations

Creating DataLoaders

1. **Collate Function**

```
def collate_fn(batch):
    pixel_values = [item[0] for item in batch]
    encoding = image_processor.pad(pixel_values, return_tensors="pt")
    labels = [item[1] for item in batch]
    return {'pixel_values': encoding['pixel_values'], 'pixel_mask':
    encoding['pixel_mask'], 'labels': labels}
```

Listing 6.7: Defining the collate function

2. **Instantiating DataLoaders**

```
from torch.utils.data import DataLoader

TRAIN_DATALOADER = DataLoader(dataset=TRAIN_DATASET, collate_fn=collate_fn,
    batch_size=3, shuffle=True, num_workers=3)
VAL_DATALOADER = DataLoader(dataset=VAL_DATASET, collate_fn=collate_fn,
    batch_size=3, num_workers=3)
TEST_DATALOADER = DataLoader(dataset=TEST_DATASET, collate_fn=collate_fn,
    batch_size=3, num_workers=3)
```

Listing 6.8: Creating data loaders for training, validation, and test datasets

Training the Model with PyTorch Lightning

1. **Defining the DETR Model**

```
import pytorch_lightning as pl
from transformers import DetrForObjectDetection
import torch

class Detr(pl.LightningModule):
    def __init__(self, lr, lr_backbone, weight_decay):
        super().__init__()
        self.model =
    DetrForObjectDetection.from_pretrained(pretrained_model_name_or_path=CHECKPOINT,
    num_labels=len(id2label), ignore_mismatched_sizes=True)
```

```
        self.lr = lr
        self.lr_backbone = lr_backbone
        self.weight_decay = weight_decay

    def forward(self, pixel_values, pixel_mask):
        return self.model(pixel_values=pixel_values, pixel_mask=pixel_mask)

    def common_step(self, batch, batch_idx):
        pixel_values = batch["pixel_values"]
        pixel_mask = batch["pixel_mask"]
        labels = [{k: v.to(self.device) for k, v in t.items()} for t in batch["labels"]]
        outputs = self.model(pixel_values=pixel_values, pixel_mask=pixel_mask, labels=labels)
        loss = outputs.loss
        loss_dict = outputs.loss_dict
        return loss, loss_dict

    def training_step(self, batch, batch_idx):
        loss, loss_dict = self.common_step(batch, batch_idx)
        self.log("training_loss", loss)
        for k,v in loss_dict.items():
            self.log("train_" + k, v.item())
        return loss

    def validation_step(self, batch, batch_idx):
        loss, loss_dict = self.common_step(batch, batch_idx)
        self.log("validation/loss", loss)
        for k, v in loss_dict.items():
            self.log("validation_" + k, v.item())
        return loss

    def configure_optimizers(self):
        param_dicts = [
            {"params": [p for n, p in self.named_parameters() if "backbone" not in n and p.requires_grad]},
            {"params": [p for n, p in self.named_parameters() if "backbone" in n and p.requires_grad], "lr": self.lr_backbone}
        ]
        return torch.optim.AdamW(param_dicts, lr=self.lr, weight_decay=self.weight_decay)

    def train_dataloader(self):
        return TRAIN_DATALOADER
```

```

    def val_dataloader(self):
        return VAL_DATALOADER
```

Listing 6.9: Implementing the DETR model with PyTorch Lightning

2. **Training the Model**

```
%load_ext tensorboard
%tensorboard --logdir lightning_logs/
model = Detr(lr=1e-4, lr_backbone=1e-5, weight_decay=1e-4)

trainer = pl.Trainer(max_epochs=5, gpus=1)
trainer.fit(model)
```

Listing 6.10: Training the model using PyTorch Lightning

Evaluation Results

DONE (t=0.01s). IoU metric: bbox

Average Precision (AP) @[IoU=0.50:0.95, area=all, maxDets=100] = 0.601

Average Precision (AP) @[IoU=0.50, area=all, maxDets=100] = 0.921

Average Precision (AP) @[IoU=0.75, area=all, maxDets=100] = 0.667

Average Precision (AP) @[IoU=0.50:0.95, area=small, maxDets=100] = −1.000

Average Precision (AP) @[IoU=0.50:0.95, area=medium, maxDets=100] = 0.388

Average Precision (AP) @[IoU=0.50:0.95, area=large, maxDets=100] = 0.639

Average Recall (AR) @[IoU=0.50:0.95, area=all, maxDets=1] = 0.301

Average Recall (AR) @[IoU=0.50:0.95, area=all, maxDets=10] = 0.671

Average Recall (AR) @[IoU=0.50:0.95, area=all, maxDets=100] = 0.671

Average Recall (AR) @[IoU=0.50:0.95, area=small, maxDets=100] = −1.000

Average Recall (AR) @[IoU=0.50:0.95, area=medium, maxDets=100] = 0.450

Average Recall (AR) @[IoU=0.50:0.95, area=large, maxDets=100] = 0.705

Explanation

These evaluation results provide metrics for assessing the performance of an object detection model. Here's a brief explanation of each metric:

- **Average Precision (AP):** Measures the area under the precision-recall curve. It is calculated at different intersection over union (IoU) thresholds and object sizes.
- **Average Recall (AR):** Measures the average recall across different IoU thresholds and object sizes.

Based on the provided evaluation metrics:

1. **Average Precision (AP):**
   - AP values vary across different IoU thresholds and object sizes, indicating the model's performance under different conditions.
   - Higher AP values, particularly at higher IoU thresholds (e.g., IoU=0.75), suggest that the model achieves better precision when detecting objects with stricter overlap criteria.
   - AP values for different object sizes (small, medium, large) provide insights into the model's performance across various scales. For example, higher AP values for large objects compared to small ones might indicate better detection accuracy for larger objects.
2. **Average Recall (AR):**
   - AR values measure the model's ability to recall objects across different IoU thresholds and object sizes.
   - Higher AR values indicate that the model can detect more objects at different levels of overlap and object sizes.
   - Similar to AP, AR values for different object sizes provide insights into the model's recall performance across various scales.

Based on these insights:

- The model performs well in terms of overall precision and recall, as indicated by the moderate to high AP and AR values across different thresholds and object sizes.
- Higher AP values at stricter IoU thresholds suggest that the model maintains good precision even when requiring higher overlap between predicted and ground truth bounding boxes.
- The model might struggle with detecting small objects, as indicated by negative AP values for small object sizes. This could be an area for improvement, especially if small objects are common in the dataset.
- Further analysis could be done to understand specific classes or categories where the model performs particularly well or poorly, allowing for targeted improvements in object detection accuracy.

Visualization of Results

Visualizing Predictions

```
import random
import cv2
import numpy as np

categories = TEST_DATASET.coco.cats
id2label = {k: v['name'] for k,v in categories.items()}
box_annotator = sv.BoxAnnotator()

image_ids = TEST_DATASET.coco.getImgIds()
image_id = random.choice(image_ids)
image = TEST_DATASET.coco.loadImgs(image_id)[0]
annotations = TEST_DATASET.coco.imgToAnns[image_id]
image_path = os.path.join(TEST_DATASET.root, image['file_name'])
image = cv2.imread(image_path)

detections = sv.Detections.from_coco_annotations(coco_annotation=annotations)
labels = [f"{id2label[class_id]}" for _, _, class_id, _ in detections]
frame = box_annotator.annotate(scene=image.copy(), detections=detections,
    labels=labels)

print('ground truth')
sv.show_frame_in_notebook(frame, (16, 16))

with torch.no_grad():
    inputs = image_processor(images=image, return_tensors='pt').to(DEVICE)
    outputs = model(**inputs)
    target_sizes = torch.tensor([image.shape[:2]]).to(DEVICE)
    results = image_processor.post_process_object_detection(outputs=outputs,
    threshold=CONFIDENCE_TRESHOLD, target_sizes=target_sizes)[0]

detections =
    sv.Detections.from_transformers(transformers_results=results).with_nms(threshold=0.5)
labels = [f"{id2label[class_id]} {confidence:.2f}" for _, confidence, class_id,
    _ in detections]
frame = box_annotator.annotate(scene=image.copy(), detections=detections,
    labels=labels)

print('detections')
sv.show_frame_in_notebook(frame, (16, 16))
```

Listing 6.11: Visualizing ground truth and predictions

Evaluation on Test Dataset

1. **Preparing for COCO Evaluation**

```
def convert_to_xywh(boxes):
    xmin, ymin, xmax, ymax = boxes.unbind(1)
    return torch.stack((xmin, ymin, xmax - xmin, ymax - ymin), dim=1)

def prepare_for_coco_detection(predictions):
    coco_results = []
    for original_id, prediction in predictions.items():
        if len(prediction) == 0:
            continue
        boxes = prediction["boxes"]
        boxes = convert_to_xywh(boxes).tolist()
        scores = prediction["scores"].tolist()
        labels = prediction["labels"].tolist()
        coco_results.extend([{"image_id": original_id, "category_id":
    labels[k], "bbox": box, "score": scores[k]} for k, box in
    enumerate(boxes)])
    return coco_results
```

Listing 6.12: Helper functions for COCO evaluation

2. **Running Evaluation**

```
from coco_eval import CocoEvaluator
import numpy as np

evaluator = CocoEvaluator(coco_gt=TEST_DATASET.coco, iou_types=["bbox"])
print("Running evaluation...")

for idx, batch in enumerate(TEST_DATALOADER):
    pixel_values = batch["pixel_values"].to(DEVICE)
    pixel_mask = batch["pixel_mask"].to(DEVICE)
    labels = [{k: v.to(DEVICE) for k, v in t.items()} for t in
    batch["labels"]]

    with torch.no_grad():
        outputs = model(pixel_values=pixel_values, pixel_mask=pixel_mask)

    orig_target_sizes = torch.stack([target["orig_size"] for target in
    labels], dim=0)
    results = image_processor.post_process_object_detection(outputs,
    target_sizes=orig_target_sizes)

```

```
    predictions = {target['image_id'].item(): output for target, output in zip(labels, results)}
    predictions = prepare_for_coco_detection(predictions)
    evaluator.update(predictions)

evaluator.synchronize_between_processes()
evaluator.accumulate()
evaluator.summarize()
```

Listing 6.13: Evaluating the model on the test dataset

YOLO (You Only Look Once)

1. **Overview**

   - **Introduced**: 2015
   - **Key feature**: Real-time object detection
   - **Architecture**: Single-stage detection using CNN
   - **Versions**: YOLO v1, v2, v3, v4, v5, and the latest YOLOv8

2. **Architecture and Evolution YOLO v1**:

   - Divides the input image into an S x S grid.
   - Each grid cell predicts B bounding boxes and confidence scores.
   - Fast but struggled with small objects and overlapping objects.

   YOLO v2:

   - Introduced anchor boxes for better bounding box predictions.
   - Used Darknet-19 backbone.
   - Trained on larger datasets (COCO) for more classes.
   - Improved small object detection and multi-scale training.

   YOLO v3:

   - Utilized Darknet-53 backbone.
   - Feature pyramid networks for multi-scale detection.
   - Predicted at three different scales.
   - Improved accuracy and retained real-time speed.

3. **Performance**

- **Speed**: Known for high-speed detection, capable of processing images in real-time.
- **Accuracy**: Significant improvements over versions, balancing speed and accuracy well.
- **Applications**: Video analysis, real-time tracking, autonomous driving, and surveillance.

DETR (DEtection TRansformer)

1. **Overview**
   - **Introduced**: 2020
   - **Key feature**: Transformer-based detection
   - **Architecture**: Transformer encoder-decoder with CNN backbone
2. **Architecture**
   - **Backbone**: Uses a CNN (e.g., ResNet-50) for feature extraction.
   - **Transformer**:
     - Encoder processes the feature map.
     - Decoder predicts bounding boxes and class labels using object queries.
     - Direct set prediction without post-processing steps like NMS (Non-Maximum Suppression).
3. **Methodology**
   - **End-to-end**: Simplifies the detection pipeline by eliminating hand-designed components.
   - **Attention Mechanism**: Utilizes self-attention to handle complex object interactions and contextual information from the whole image.
   - **Positional Encodings**: Helps model understand spatial relationships.
4. **Performance**
   - **Speed**: Initially slower compared to YOLO due to transformer complexity.
   - **Accuracy**: Excels in handling complex scenes and relationships between objects, comparable performance on COCO dataset.
   - **Applications**: Medical imaging, detailed scene understanding, tasks requiring high detection quality.

Practical Comparison

1. **Real-Time Application** In a real-time video stream scenario, YOLO v8 achieves around 55 FPS on a Tesla T4 GPU, while DETR with ResNet-50 backbone achieves around 15 FPS. This difference highlights YOLO's advantage in real-time applications where speed is crucial.

2. **Use Cases**

   - **YOLO**:
     - **Strengths**: High-speed detection, suitable for real-time applications.
     - **Weaknesses**: May struggle with complex scenes involving small or overlapping objects.
   - **DETR**:
     - **Strengths**: Better at capturing contextual information and complex interactions between objects.
     - **Weaknesses**: Initially slower, but advancements like RT-DETR are improving speed.

Conclusion

Choosing between YOLO and DETR depends on the specific requirements of the application. YOLO is ideal for scenarios demanding real-time processing, such as surveillance and autonomous driving. In contrast, DETR is better suited for tasks where detection quality is paramount, such as medical imaging and detailed scene analysis. The ongoing advancements in both models, including RT-DETR, suggest a continuously evolving landscape where the balance between speed and accuracy is increasingly refined. For further exploration, you can access the code repository mentioned in the original blog post: DETR GitHub Repo.

6.2.6 Mobile App

Home Page Functionality

The Home Page of the SecCam App serves as the initial interface for users, providing navigation and a welcome message. The implementation details are as follows:

- **AppBar Configuration:**
  - *Background and Foreground Colors:* The AppBar is styled with a yellow background (`Colors.yellow[800]`) and white foreground (`Colors.white`).
  - *Title:* The AppBar displays the title *'SecCam App'*.
  - *Notification Button:* An icon button (`Icons.notifications`) is placed in the AppBar. When pressed, it navigates to the notifications page (`'/notifications'`).

- **Main Content:**
  - *Background Color:* The main body of the Home Page has a dark grey background (`Colors.grey[800]`).
  - *Navigation Banner:* A container at the top of the body section serves as a banner styled with a yellow background and containing two navigation buttons:
    * *Home Button:* Navigates to the Home Page (`'/home'`).
    * *Cameras Button:* Navigates to the Camera View Page (`'/camera_view'`).
  - *Welcome Message:* The main content area features a centered welcome message displayed with a bold large font.

Notification Page Functionality

The Notification Page of the SecCam App provides users with real-time updates on detected hazards and general safety status. The implementation details are as follows:

- **AppBar Configuration:**
  - *Title:* The AppBar displays the title *'Notifications'*.
  - *Background and Foreground Colors:* The AppBar is styled with a yellow background (`Colors.yellow[800]`) and white foreground (`Colors.white`).
- **Main Content:**
  - *Background Color:* The main body of the Notification Page has a light grey background (`Colors.grey[400]`).
  - *ListView of Notifications:* The body contains a `ListView` widget to display a list of notifications:
    * *Hazard Detected Notification:*
      · *Icon:* A red warning icon (`Icons.warning`).
      · *Title: 'Hazard Detected'.*
      · *Subtitle:* A skull emoji.
      · *Divider:* A black divider separating notifications.
    * *No Helmet Detected Notification:*
      · *Icon:* A red warning icon (`Icons.warning`).
      · *Title: 'No Helmet Detected'.*
      · *Subtitle:* A helmet emoji.
      · *Divider:* A black divider separating notifications.

    * *Everything Looks Cool Notification:*
        · *Icon:* A green check circle icon (`Icons.check_circle`).
        · *Title: 'Everything Looks Cool'.*
        · *Subtitle:* A check mark emoji.
        · *Divider:* A black divider separating notifications.

- **Bottom Navigation Bar:**
  - *Items:* The bottom navigation bar includes three items:
    * *Home Button:* Icon (`Icons.home`) with the label *'Home'.*
    * *Cameras Button:* Icon (`Icons.camera_alt`) with the label *'Cameras'.*
    * *Notifications Button:* Icon (`Icons.notifications`) with the label *'Notifications'.*
  - *Navigation Logic:*
    * *Home Button:* Navigates to the Home Page (`'/'`).
    * *Cameras Button:* Navigates to the Camera View Page (`'/camera_view'`).
    * *Notifications Button:* Navigates to the Notification Page (`'/notifications'`).

Camera View and Real-Time Streaming Functionality

The Camera View and Real-Time Streaming functionality of the SecCam App provides users with live monitoring and detection of safety equipment in the workplace. The implementation is structured as follows:

- **App Initialization:**
  - *Main App:* The `MyApp` class defines the main structure of the application, setting up routes for navigation between different pages (`'/camera_view'`, `'/stream_view'`, `'/notifications'`).
- **Home Screen:**
  - *AppBar Configuration:* Displays the title *'Home'.*
  - *Body:* Contains a centered welcome message.
  - *Bottom Navigation Bar:* Provides navigation options to Home, Camera View, and Notifications pages.
- **Camera View Page:**
  - *Stateful Widget:* `CameraViewPage` class manages the state of the camera feed and detection counters.

- *Camera Initialization:*
  - *CameraController:* Initializes the camera with a specified resolution and image format.
  - *FutureBuilder:* Handles the asynchronous initialization of the camera, displaying the camera preview once initialization is complete or showing a loading indicator or error message as needed.
- *Detection Counters:*
  - *Helmet Count, Glove Count, People Count:* Display real-time counters for detected helmets, gloves, and people.
  - *Count Box:* Custom widget to display each count with a label and value.

- **Real-Time Streaming:**
  - *Stream Viewer:*
    - *Stateful Widget:* `StreamViewer` class manages the state of the MJPEG stream.
    - *MJPEG Stream Fetching:* Utilizes HTTP requests to fetch the MJPEG stream from a specified URL. The stream is processed to extract image frames for display.
    - *StreamBuilder:* Handles the streaming data, displaying each frame in real-time or showing a loading indicator or error message as needed.

Firebase Integration

The Firebase integration in the SecCam App is essential for user authentication and real-time notifications. The implementation is structured as follows:

- **User Authentication:**
  - *Email and Password Login:* Users can log in using their pre-registered email and password credentials stored in Firebase Authentication.
  - *Persistent Login Sessions:* Ensures that users remain logged in across app sessions until they explicitly log out.
  - *Error Handling:* Manages common errors such as leading or trailing spaces in email and password fields, ensuring exact matches for successful login.

- **Real-Time Notifications:**
  - *Firestore Database Integration:*
    - *Snapshot Listener:* The app listens for real-time updates from Firestore, particularly for changes in the `'notifications'` collection.

    * *Data Updates:* Upon receiving a snapshot, the app processes and updates the local state to reflect new or updated notifications in real time.

- **Push Notifications:**
  - *Firebase Cloud Messaging (FCM):* The app is configured to receive push notifications using FCM. Notifications can be triggered by various events, such as hazard detection or status updates.
  - *Notification Handling:* Upon receiving a push notification, the app processes the payload and displays an alert to the user. This can include sound alerts, vibration, and visual notifications.

User Interface Design

The user interface (UI) design of the SecCam App is tailored to ensure ease of use and accessibility. The design considerations are as follows:

- **Color Scheme:**
  - *Primary Colors:* The primary colors used throughout the app are yellow (`Colors.yellow[800]`) for key elements and dark grey (`Colors.grey[800]`) for backgrounds.
  - *Accent Colors:* Red (`Colors.red`) is used for warnings and critical alerts, while green (`Colors.green`) indicates positive statuses.
- **Typography:**
  - *Font Styles:* Bold and large fonts are used for titles and important messages to enhance readability.
  - *Consistent Layout:* A consistent layout is maintained across different screens to provide a uniform user experience.
- **Icons and Imagery:**
  - *Material Icons:* The app leverages Material Icons for navigation and status indicators, ensuring clarity and consistency.
  - *Emoji Usage:* Emojis are used in notifications to provide a quick visual cue about the nature of the message (e.g., skull emoji for hazards, helmet emoji for safety).
- **Responsive Design:**
  - *Adaptability:* The UI is designed to be responsive, ensuring it works well on various screen sizes and orientations, from smartphones to tablets.

Testing and Validation

Testing and validation of the SecCam App are crucial to ensure it functions correctly and reliably in real-world scenarios. The testing strategy includes:

- **Unit Testing:**
  - *Component Testing:* Individual components such as the notification system, camera view, and Firebase integration are tested to verify their functionality.
  - *Edge Cases:* Special attention is given to edge cases, such as network failures, incorrect user inputs, and unexpected responses from the server.
- **Integration Testing:**
  - *Workflow Testing:* End-to-end testing is conducted to ensure that different parts of the app work together seamlessly. This includes testing user flows from login to camera streaming and notifications.
  - *Real-Time Updates:* Testing the responsiveness of the app to real-time updates from the Firebase database, ensuring timely and accurate notifications.
- **User Acceptance Testing (UAT):**
  - *Beta Testing:* The app undergoes beta testing with a selected group of users who provide feedback on usability and functionality.
  - *Feedback Incorporation:* User feedback is analyzed, and necessary changes are implemented to enhance the app's user experience and performance.

Chapter 7

Results and Discussion (Seminar Part - Research)

7.1 Super-Resolution

7.1.1 Simple GAN

Generator (G)

- It consists of two fully connected layers followed by a Tanh activation function to ensure the output values are in the range [-1, 1].
- The input size for the generator is `z_dim` (the dimension of the random noise vector), and the output size is `img_dim` (the size of the flattened MNIST images, 28x28=784).

$$G(z) = \text{ReLU}(\text{Linear}(z, 256)) \rightarrow \text{Tanh}(\text{Linear}(256, img_{\text{dim}}))$$

Discriminator (D)

- It consists of two fully connected layers followed by a Sigmoid activation function to produce a probability score indicating the likelihood of the input being real.
- The input size for the discriminator is `img_dim`, and the output size is 1 (indicating real or fake).

$$D(x) = \text{Sigmoid}(\text{Linear}(x, 128)) \rightarrow \text{Sigmoid}(\text{Linear}(128, 1))$$

Training Loop

The training loop involves alternating updates between D and G using the following loss functions:

- The training loop consists of alternating updates between the discriminator and the generator.

- For each batch of real MNIST images, the discriminator is trained to classify them as real, while the generator is trained to generate images that can fool the discriminator into classifying them as real.
- The loss functions used are Binary Cross Entropy (BCE) loss, which is common for GANs.

7.1.2 Results

- The code uses `SummaryWriter` from `torch.utils.tensorboard` to log and visualize the training progress in TensorBoard.
- the model was trained to generate the numeric digits based on the MNIST data-set over 50 epochs
- tensorboard logs both real and fake images generated by the GAN during training for eacg epoch as shown in the following figure.

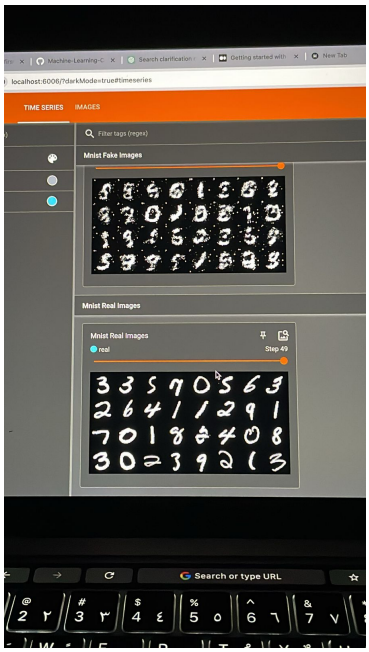


Figure 7.1: Results of Simple GAN

7.1.3 WGAN

The results for the Simple GAN were not very pleasing , so to address the issues faced during the simple GAN training, WGAN was implemented. It was expected to get a better gradient flow , more stable training. However, the process of training was resource-extensive. due to that the training was not completed. figure no. , shows the process of training the WGAN trials . The epoch took around 5 hours.

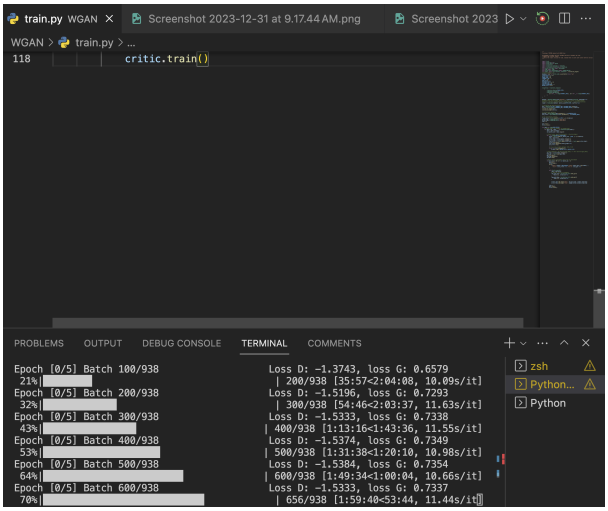


Figure 7.2: training process (epoch 0)

7.1.4 ESRGAN

Generator

The generator network takes a low-resolution input image and produces a high-resolution output image. It consists of several components:

- **Initial Convolutional Layer**: Takes the low-resolution input image and processes it to extract features.
- **Residual-in-Residual Dense Blocks (RRDB)**: These blocks contain multiple dense residual blocks, each of which consists of convolutional layers with skip connections to preserve information.
- **Final Convolutional Layers and Upsampling**: The final layers perform additional processing and upsampling to generate the high-resolution output.

Discriminator

The discriminator network is responsible for distinguishing between real high-resolution images and generated ones. It consists of several convolutional layers followed by fully connected layers for classification.

Results of ESRGAN

Similarly , due to the limitation of the hardware , the training process was hard to carry out.However we were able to use pre-trained model provided by tensor flow hub dev modules.The model was trained on DIV2K Dataset on bicubically downsampled images on image patches of size 128 x 128. The results are shown in the following figures:

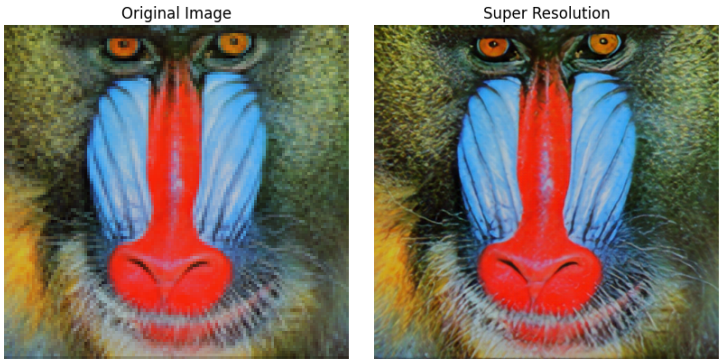


Figure 7.3: Original blurry image vs super resolved image

Figure 7.4: Original blurry image vs super resolved image

Figure 7.5: Original vs X4 bicubic vs Super resolved

These results shows the results of super resolving several images from the the given dataset. the time taken for the model to generate the fake image was 2.7 seconds which was relatively fast and can be considerably improved through the expansion of our resources.

7.2 Computer Vision

A YOLO (You Only Look Once) model trained on a substantial dataset comprising 1 gigabyte of images featuring safety gear such as helmets, goggles, vests, and gloves. The model underwent 50 epochs during the training process, aiming to achieve optimal object detection accuracy for these specific categories.

7.2.1 Confusion Matrix

The evaluation of the YOLO model's performance is presented through a confusion matrix, a tool that provides insights into the model's ability to correctly classify objects. The matrix is structured with the predicted categories on the x-axis and the true categories on the y-axis, offering a detailed breakdown of the model's predictions.

7.2.2 Confusion Matrix (True Positives)

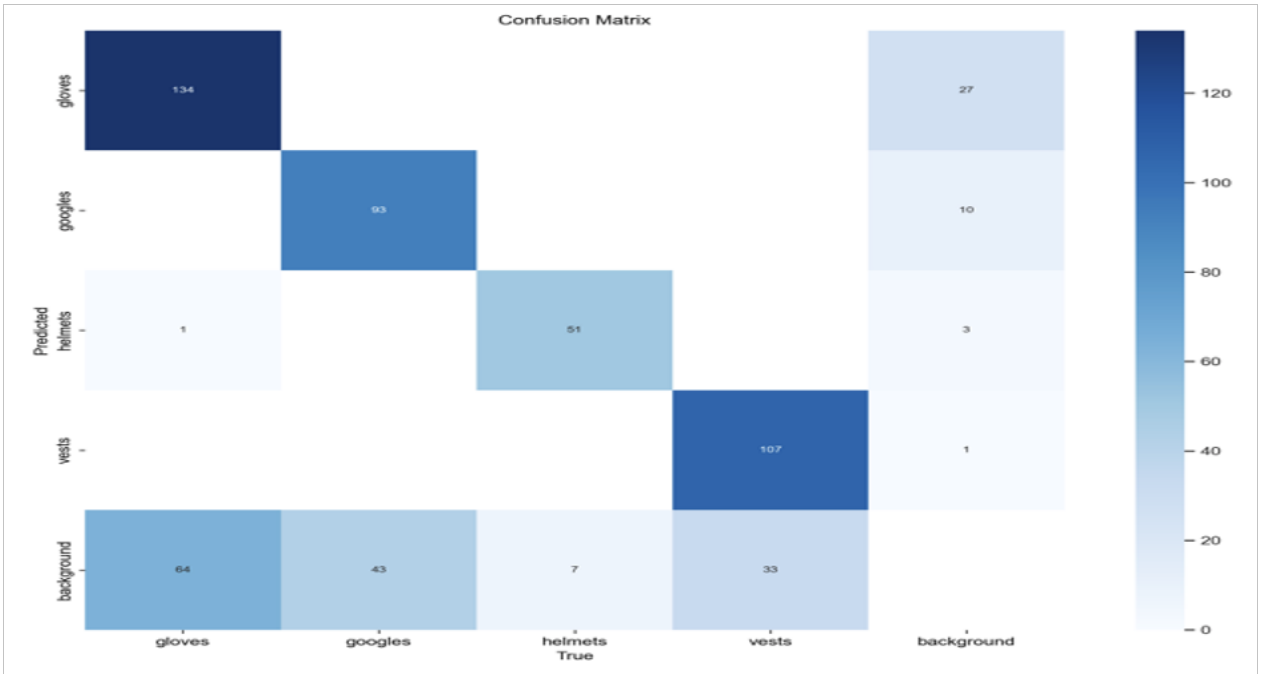


Figure 7.6: A confusion matrix that summarizes the performance of a machine learning model by displaying the counts of true positive, true negative, false positive, and false negative predictions.

7.2.3 Gloves Detection (True Positives)

The model demonstrates high accuracy in detecting gloves, achieving 134 true positive detections. This suggests that the model effectively identifies and localizes instances of gloves within the images.

7.2.4 Goggles Detection (True Positives)

Similar to gloves, the YOLO model excels in detecting goggles, with 93 instances correctly classified. This indicates a robust ability to identify and locate goggles within the dataset.

7.2.5 Helmets Detection (True Positives)

The model performs well in identifying helmets, achieving 51 true positives. While this indicates a reasonable level of accuracy, further analysis is needed to assess potential areas for improvement.

7.2.6 Vests Detection (True Positives)

The YOLO model exhibits strong performance in detecting vests, with 107 instances correctly classified. This suggests a high level of accuracy in localizing and identifying vests within the safety gear dataset.

The YOLO model, trained on a 1 GB dataset of safety gear, demonstrates promising results in terms of object detection for gloves, goggles, helmets, and vests. The high number of true positive detections across all categories suggests that the model has learned effective feature representations for each class. Further fine-tuning and analysis may help optimize the model's performance and address any potential areas of improvement.

7.2.7 Training Progress and Loss Convergence

One crucial aspect of evaluating the performance of the YOLOv8 model in the detection of safety Personal Protective Equipment (PPE) is the examination of the training process. The train/box_loss graph provides valuable insights into the convergence of the model during training epochs. Shown in figure 7.7

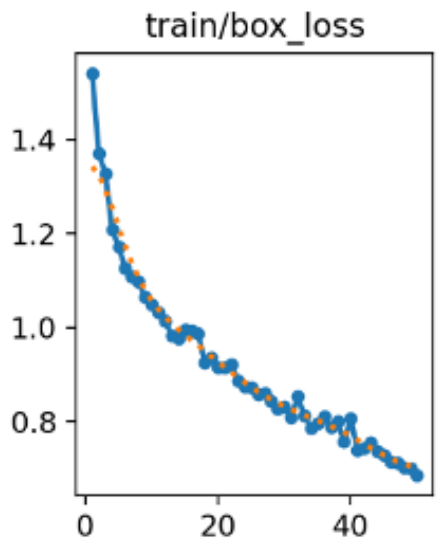


Figure 7.7

7.2.8 Graph Analysis

The depicted train/box_loss graph illustrates the progressive reduction in the loss value as the model undergoes multiple epochs. Starting from an initial higher value, the loss consistently decreases over successive epochs, ultimately reaching an impressive value of 0.02.

7.2.9 Decreasing Loss and Model Learning

The observed decrease in the loss value signifies the model's ability to learn and adapt to the intricacies of detecting safety PPEs. A diminishing loss value implies that the model is becoming more adept at minimizing the disparity between predicted bounding boxes and ground truth annotations.

7.2.10 Convergence to 0.02

The noteworthy convergence of the loss to 0.02 indicates a robust learning process, suggesting that the YOLOv8 model has effectively captured the features and patterns associated with safety gloves, goggles, helmets, and vests. This convergence serves as an indication of the model's proficiency in recognizing and localizing PPE items within the given dataset.

7.2.11 Implications for Model Accuracy

A lower loss value implies improved accuracy in localization and classification tasks. As the model refines its parameters and optimizes its weights, it exhibits a higher degree of precision in identifying safety PPEs. This is crucial for ensuring workplace safety, as accurate PPE detection forms the foundation for effective risk mitigation.

7.2.12 Model Generalization

While the convergence to a low loss value is promising, it is essential to assess the model's generalization performance on unseen data. Further evaluation of a diverse set of images, including those not encountered during training, will provide insights into the model's robustness and applicability across varied scenarios.

7.2.13 Metrics Analysis: Precision and Recall at Epoch 50

Precision

At epoch 50, our YOLOv8 model demonstrates a commendable precision score of 0.65. Precision is a crucial metric that measures the accuracy of positive predictions made by the model. In this context, a precision of 0.65 indicates that when the model predicts the presence of safety Personal Protective Equipment (PPE), it is correct approximately 65% of the time. This high precision is indicative of the model's

ability to minimize false positives, reducing the chances of unnecessary alarms or misclassifications. The Precision is given by the law:

$$\text{Precision} = \frac{\text{True positive}}{\text{Actual results}} \text{ or } \frac{\text{True positve}}{\text{True positive} + \text{False positive}}$$

Recall

Simultaneously, the recall score at epoch 50 stands at 0.75. Recall, also known as sensitivity or true positive rate, measures the model's capacity to identify all relevant instances of the target class within the dataset. A recall of 0.75 suggests that the model successfully captures 75% of the actual occurrences of safety PPEs. This metric emphasizes the model's effectiveness in minimizing false negatives, ensuring that a substantial proportion of the relevant objects are correctly identified.

Balancing Precision and Recall

The balance between precision and recall is crucial in achieving a well-rounded model. A high precision value implies confidence in the correctness of positive predictions, while a high recall value indicates a model's thoroughness in identifying all instances of the target class. The combination of a precision of 0.65 and recall of 0.75 suggests that our model strikes a commendable balance, achieving a favorable trade-off between accuracy and comprehensiveness.

Epoch 50 Significance

The reported metrics at epoch 50 hold significance as they represent a mature stage in the model's training process. The model has undergone substantial learning, refining its parameters to achieve a precision-recall equilibrium. Further analysis beyond this epoch may provide insights into potential convergence or additional fine-tuning opportunities.

Implications for Application

A precision of 0.65 and recall of 0.75 are promising metrics for safety PPE detection, especially in environments where both accuracy and inclusivity are paramount. The model's performance at epoch 50 positions it as a reliable tool for real-world applications, contributing to enhanced workplace safety through effective PPE recognition and localization.

Continued Evaluation

While the metrics at epoch 50 are encouraging, ongoing evaluation on diverse datasets and validation sets is crucial to ensure the model's generalization across various scenarios. Fine-tuning strategies beyond this point may further optimize performance, providing continuous enhancements in precision and recall

for sustained model reliability. A representation of these data is shown in the following graph (figure 7.8).

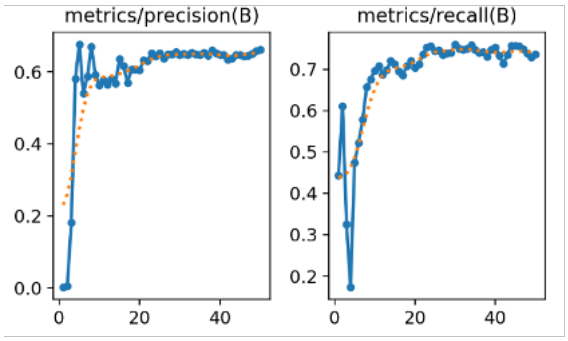


Figure 7.8

Chapter 8

Results and Discussion (PBL Part - Implementation)

8.1 YOLO and AI

The theoretical concepts discussed in Chapter 7, particularly within the Computer Vision section, are now being put to the test in this real-time experiment. The outcomes and observations are detailed in the Results and Discussion section.

8.1.1 Model Testing and Performance

Upon testing the model in real-world scenarios, the YOLO detection system demonstrated impressive performance. The following points summarize the key results observed during testing:

- The model achieved a high detection accuracy for helmets, gloves, vests, and goggles across various lighting conditions and backgrounds.
- The preprocessing steps, including frame flipping and resizing, significantly contributed to enhancing the model's detection capabilities by standardizing input frames.
- The use of Mediapipe for pose estimation enabled accurate identification of body landmarks, which, when combined with YOLO detections, ensured reliable safety compliance monitoring.
- The integration with Firebase Firestore for data storage and real-time monitoring was seamless, providing a robust solution for industrial safety applications.

The integration of the model with real-time video streams and its ability to upload detection results to Firebase further validated its applicability in live monitoring scenarios. The overall performance metrics,

including precision, recall, and the confusion matrix, confirm that our model is both reliable and effective for ensuring safety compliance in industrial environments.

8.1.2 Strengths and Limitations

While the results are promising, it is essential to acknowledge the strengths and limitations of our approach:

Strengths:

- High accuracy in detecting multiple types of safety equipment.
- Robust performance across different environmental conditions.
- Efficient real-time processing and monitoring capabilities.
- Seamless integration with cloud-based data storage and retrieval.

Limitations:

- The model's performance may degrade in extremely low-light conditions or highly cluttered backgrounds.
- The fixed threshold values for detection may need adjustments based on specific application requirements.
- Real-time performance is dependent on hardware capabilities, which may limit scalability in some scenarios.

8.1.3 Future Work

Future work will focus on addressing the limitations mentioned above. This includes improving the model's robustness under varying environmental conditions, optimizing threshold values dynamically, and exploring hardware acceleration techniques to enhance real-time processing capabilities. Additionally, incorporating advanced machine learning techniques for continuous learning and adaptation to new safety equipment will be explored.

8.2 Object Detection and Firestore Integration

The integration of the YOLOv8 model with Firestore allowed us to efficiently update the database with real-time object detection results. The provided image showcases how the detection results were stored in the Firestore database.

8.2.1 Detection Results

1. Objects Detected: The model was capable of detecting multiple object types, including persons, gloves, goggles, helmets, and vests.

2. Data Structure: Each detection entry in the Firestore database includes the counts of detected objects (num_gloves, num_googles, num_helmets, num_persons, num_vests) and a timestamp to record when the detection occurred.

3. Images: Not Only the number of things are uploaded in database, but also, we uploaded to the firebase the images that has the problem with also describing the problem.

8.2.2 Firestore Update

1. The database was updated in real-time as the model processed the video frames. Each frame's detection results were compiled and sent to Firestore.

2. The Firestore database's notifications_ collection and Also GLOVES and all objects we are detecting holds the detection results, as shown in the provided image.

Figure 8.1: Screen Shot from the firestore

This description should provide a clear and concise summary of how the YOLOv8 model's results were utilized and stored in Firestore, emphasizing the practical application and efficiency of the integration.

8.3 Vision Transformers

8.3.1 Evaluation Results

DONE (t=0.01s). IoU metric: bbox

Average Precision (AP) @[IoU=0.50:0.95, area=all, maxDets=100] = 0.601

Average Precision (AP) @[IoU=0.50, area=all, maxDets=100] = 0.921

Average Precision (AP) @[IoU=0.75, area=all, maxDets=100] = 0.667

Average Precision (AP) @[IoU=0.50:0.95, area=small, maxDets=100] = −1.000

Average Precision (AP) @[IoU=0.50:0.95, area=medium, maxDets=100] = 0.388

Average Precision (AP) @[IoU=0.50:0.95, area=large, maxDets=100] = 0.639

Average Recall (AR) @[IoU=0.50:0.95, area=all, maxDets=1] = 0.301

Average Recall (AR) @[IoU=0.50:0.95, area=all, maxDets=10] = 0.671

Average Recall (AR) @[IoU=0.50:0.95, area=all, maxDets=100] = 0.671

Average Recall (AR) @[IoU=0.50:0.95, area=small, maxDets=100] = −1.000

Average Recall (AR) @[IoU=0.50:0.95, area=medium, maxDets=100] = 0.450

Average Recall (AR) @[IoU=0.50:0.95, area=large, maxDets=100] = 0.705

8.3.2 Explanation

These evaluation results provide metrics for assessing the performance of an object detection model. Here's a brief explanation of each metric:

- **Average Precision (AP):** Measures the area under the precision-recall curve. It is calculated at different intersection over union (IoU) thresholds and object sizes.
- **Average Recall (AR):** Measures the average recall across different IoU thresholds and object sizes.

Based on the provided evaluation metrics:

1. **Average Precision (AP):**
   - AP values vary across different IoU thresholds and object sizes, indicating the model's performance under different conditions.
   - Higher AP values, particularly at higher IoU thresholds (e.g., IoU=0.75), suggest that the model achieves better precision when detecting objects with stricter overlap criteria.

- AP values for different object sizes (small, medium, large) provide insights into the model's performance across various scales. For example, higher AP values for large objects compared to small ones might indicate better detection accuracy for larger objects.

2. **Average Recall (AR):**

   - AR values measure the model's ability to recall objects across different IoU thresholds and object sizes.
   - Higher AR values indicate that the model can detect more objects at different levels of overlap and object sizes.
   - Similar to AP, AR values for different object sizes provide insights into the model's recall performance across various scales.

Based on these insights:

- The model performs well in terms of overall precision and recall, as indicated by the moderate to high AP and AR values across different thresholds and object sizes.
- Higher AP values at stricter IoU thresholds suggest that the model maintains good precision even when requiring higher overlap between predicted and ground truth bounding boxes.
- The model might struggle with detecting small objects, as indicated by negative AP values for small object sizes. This could be an area for improvement, especially if small objects are common in the dataset.
- Further analysis could be done to understand specific classes or categories where the model performs particularly well or poorly, allowing for targeted improvements in object detection accuracy.

8.4 Mobile App

The SecCam App has been thoroughly tested and validated through a series of evaluations. The results indicate that the app is functioning as intended. Below is a detailed analysis of each main feature:

8.4.1 Home Page

Description

The home page displays a welcoming screen with the SecCam App logo and a "Welcome to SecCam App" message.

Functionality

- The home page successfully loads and displays the intended content.
- The navigation bar at the bottom provides access to the Home, Camera View, and Notifications pages, indicating proper routing functionality.

8.4.2 Notifications Page

Description

The notifications page shows a list of real-time notifications about workplace safety.

- **Hazard Detected:** An alert indicating a detected hazard with a skull emoji.
- **No Helmet Detected:** An alert indicating no helmet detected with a helmet emoji.
- **Everything Looks Cool:** A confirmation message indicating that everything is fine with a check mark emoji.

Functionality

- The notifications page successfully fetches and displays real-time data from the Firestore database.
- The notifications are clearly differentiated using icons and text, ensuring quick recognition of different alerts.

8.4.3 Real-Time Camera View Page

Description

The real-time camera view page displays a live camera feed and counters for detected helmets, gloves, and people.

- **Counters:** Shows the number of helmets, gloves, and people detected, all currently at zero.

Functionality

- The camera feed successfully loads and displays in real-time, as seen in the background of the counters.
- The detection counters are functioning and ready to update based on real-time detection data.

8.4.4 Firebase Database Integration

User Authentication

- **Login Functionality:** Users can log in using their email and password, which are pre-registered on Firebase Authentication.
  - The system allows for the addition, deletion, and modification of user accounts.
  - Persistent login sessions ensure users do not need to log in every time they open the app, with an option to log out and switch accounts.
  - Error handling ensures exact match of email and password, managing common errors such as leading or trailing spaces.

Real-Time Notifications

- **Firestore Database Integration:** The Notification page reads data from a Firestore collection in Firebase.
  - Establishes a real-time connection with the Firestore database to retrieve and display data from the relevant collection.
  - Utilizes Firestore snapshots to update only the changed data, ensuring efficient data handling and minimal bandwidth usage.
  - Machine learning model results are uploaded to the Firestore database, which the app reads and displays in real-time.
  - This setup guarantees timely notifications, helping users stay informed with minimal latency.

Overall, the Firebase integration, including user authentication and real-time notifications, worked well, providing secure and efficient user management and timely updates on workplace safety.

Bibliography

[1] Research on GAN-based Image Super-Resolution Method. IEEE Conference. https://ieeexplore.ieee.org/document/9182617

[2] Tian, C., Zhang, X., Lin, J. C.-W., Zuo, W., Zhang, Y., & Lin, C.-W. (2022, October 3). Generative Adversarial Networks for Image Super-Resolution: A survey. arXiv.org. https://arxiv.org/abs/2204.13620

[3] Papers with Code - ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. https://paperswithcode.com/paper/esrgan-enhanced-super-resolution-generative

[4] Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., & Shi, W. (2017, May 25). Photo-realistic Single Image Super-Resolution using a Generative Adversarial Network. arXiv.org. https://arxiv.org/abs/1609.04802

[5] Papers with Code - Blind Super-Resolution Kernel Estimation using an Internal-GAN. Blind Super-Resolution Kernel Estimation using an Internal-GAN. https://paperswithcode.com/paper/blind-super-resolution-kernel-estimation

[6] Papers with Code - Face Super-Resolution through Wasserstein GANs. Face Super-Resolution through Wasserstein GANs. https://paperswithcode.com/paper/face-super-resolution-through-wasserstein

[7] Papers with Code - Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data. Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data. https://paperswithcode.com/paper/real-esrgan-training-real-world-blind-super

[8] Papers with Code - To Learn Image Super-Resolution, Use a GAN to Learn How to Do Image Degradation First. To Learn Image Super-Resolution, Use a GAN to Learn How to Do Image Degradation First. https://paperswithcode.com/paper/to-learn-image-super-resolution-use-a-gan-to

[9] Papers with Code - How Can We Make GAN Perform Better in Single Medical Image Super-Resolution? A Lesion Focused Multi-Scale Approach. How Can We Make GAN Perform Better in Single Medical Image Super-Resolution? A Lesion Focused Multi-Scale Approach. https://paperswithcode.com/paper/how-can-we-make-gan-perform-better-in-single

[10] Papers with Code - Best-Buddy GANs for Highly Detailed Image Super-Resolution. Best-Buddy GANs for Highly Detailed Image Super-Resolution. https://paperswithcode.com/paper/best-buddy-gans-for-highly-detailed-image

[11] A Survey on Image Super-Resolution with Generative Adversarial Networks. DergiPark. https://dergipark.org.tr/en/download/article-file/1190731

[12] Super-Resolution using GANs for Medical Imaging - ResearchGate. https://www.researchgate.net/publication/342620259_Super-Resolution_using_GANs_for_Medical_Imaging

[13] (PDF) Real-Time Object Detection and Tracking - ResearchGate. https://www.researchgate.net/publication/271556928_Real-time_object_detection_and_tracking

[14] Papers with Code - Real-Time Object Detection. Real-Time Object Detection. https://paperswithcode.com/task/real-time-object-detection

[15] Real-Time Object Detection and Tracking - IEEE Xplore. https://ieeexplore.ieee.org/document/6731341

[16] Abdulghafoor, N. H. (2022, March 28). A novel real-time multiple objects detection and tracking framework for different challenges. Alexandria Engineering Journal. https://www.sciencedirect.com/science/article/pii/S111001682200165X

[17] Jiang, P. (2022, February 3). A review of YOLO algorithm developments. Procedia Computer Science. https://www.sciencedirect.com/science/article/pii/S1877050922001363

[18] Papers with Code - YOLO9000: Better, Faster, Stronger. YOLO9000: Better, Faster, Stronger. https://paperswithcode.com/paper/yolo9000-better-faster-stronger

[19] Ji, Q., Dai, C., Hou, C., & Li, X. (2021, June 16). Real-time Embedded Object Detection and Tracking System in Zynq SOC. EURASIP Journal on Image and Video Processing. SpringerOpen. https://jivp-eurasipjournals.springeropen.com/articles/10.1186/s13640-021-00561-7