Beyond the Pixels:

Emerging Computer Vision Research Topics

Beyond the Pixels: Emerging Computer Vision Research Topics serves as a comprehensive class book exploring the cutting-edge advancements shaping the future of computer vision. From state-of-the-art algorithms to groundbreaking applications, it examines how machines are learning to perceive, understand, and interpret the visual world with unprecedented accuracy. This class book did a preliminary survey of what's possible in this rapidly evolving field. Perfect for students, researchers, and enthusiasts, it provides both guidance and inspiration for those eager to explore the transformative power of computer vision.

Computer Vision Fall 2024 Class Book

Authors: Mount Holyoke College COMSC341-CV (Fall 2024) Students

Editors: Sulagna Saha (Rasha), Melody Su

December 31, 2024

Contents

Cat vs. Cat Loaf Image Classification Using Convolutional Neural Networks

Farzana Chowdhury

Abstract—**This paper investigates the classification of images into two distinct categories: "Cat" and "Cat Loaf," using Convolutional Neural Networks (CNNs). The project is inspired by the nuanced challenge of recognizing animal postures, specifically differentiating traditional cat poses from loaf-like compact positions. The dataset contains 646 RGB images, evenly split between the two classes. Images were preprocessed through resizing, normalization, and data augmentation to enhance generalization. The CNN architecture employed consists of convolutional, max-pooling, and dense layers, achieving a validation accuracy of approximately 66%. This project highlights the potential of CNNs in solving posture-based classification tasks, emphasizing the importance of feature extraction and augmentation in improving model performance. The study also discusses challenges such as subtle posture variations and dataset limitations, proposing future improvements to address these issues.**

I. Introduction

Image classification is one of the most fundamental tasks in computer vision, enabling machines to understand, categorize, and interpret visual data. The process involves assigning an image to one of several predefined categories by analyzing its visual patterns, textures, and features. Applications of image classification span a wide range of domains, including medical imaging [1], autonomous vehicles [2], and wildlife monitoring [4].

Traditional image classification tasks often involve clearly distinct object categories, such as differentiating between cats and dogs, cars and bicycles, or apples and oranges. In such cases, the classifier leverages high-level visual differences to distinguish one class from another. However, in certain scenarios, the challenge lies in detecting subtle differences within the same object class. This is particularly true for tasks requiring the identification of variations in posture, expression, or fine-grained details [6].

This project investigates such a nuanced classification challenge: distinguishing between images of cats and cats in a loaf-like posture, colloquially known as "cat loaves." Unlike traditional classification tasks, which rely on obvious differences between objects, this problem demands the model to learn subtle variations in body posture and spatial structure. For example, a cat loaf is characterized by a compact, tucked-in posture with legs hidden beneath the body, creating a loaf-like silhouette. In contrast, traditional cat postures often involve more visible legs and an extended body structure.

The "Cat vs. Cat Loaf" classification problem serves as a unique testbed for exploring the effectiveness of advanced feature extraction techniques in image classification. It demonstrates the capability of convolutional neural networks (CNNs) to learn hierarchical features, ranging from low-level edges and textures to high-level patterns, such as posture recognition [1]. Moreover, this challenge highlights the importance of leveraging preprocessing techniques and data augmentation [5] to create a robust model that can generalize well despite the inherent visual similarities between the two classes.

The significance of this study extends beyond its playful theme. Accurate recognition of animal postures has broader implications for fields like veterinary science, where posture analysis can help identify health issues in pets, and wildlife monitoring, where automated systems can classify animal behaviors in natural habitats [4]. This project also serves as a foundation for further research into nuanced classification tasks, showcasing how computer vision can handle complex and subtle distinctions.

By developing and evaluating a convolutional neural network for the "Cat vs. Cat Loaf" classification task, this project aims to contribute to the growing field of fine-grained image classification. It explores the potential of deep learning to handle subtle, posture-based distinctions, paving the way for applications in specialized computer vision problems. Through this study, the objective is not only to achieve high classification accuracy but also to understand the strengths and limitations of current techniques in addressing such nuanced challenges.

A. Inspiration for the Project

The inspiration for this project stems from a fascination with how computer vision models can perceive subtle visual distinctions. While cats in loaf postures are a lighthearted subject, the task has deeper implications for understanding pose estimation, animal behavior recognition, and specialized computer vision applications. Additionally, this project aligns with the broader goal of building models that can handle nuanced classification tasks, which are critical in fields such as medical imaging and wildlife monitoring.

The challenge of distinguishing between "Cat" and "Cat Loaf" images is both technically interesting and illustrative of the capability of CNNs to learn fine-grained features. This work serves as a foundation for understanding how neural networks can tackle similar problems in other domains.

II. Background: Image Classification and CNNs

A. Image Classification

Image classification involves assigning a label to an image based on its content. It is mathematically defined as:

$$f(I) \to y \quad \text{where } y \in \{C_1, C_2, \dots, C_n\}$$

Here, f represents the classification function, I is the input image, and y is the predicted label. In this project, $n = 2$, with C_1 = Cat and C_2 = Cat Loaf. Unlike conventional classification tasks, this problem requires the model to identify subtle features that differentiate posture rather than object identity.

B. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep learning architectures specifically designed to process and analyze image data. They excel at automatically learning hierarchical features directly from raw pixel inputs, making them a cornerstone of modern computer vision tasks. CNNs are inspired by the biological processes of the visual cortex, where neurons are sensitive to specific regions of an image, enabling localized feature detection.

A CNN processes image data through a series of layers that extract increasingly complex features. The architecture typically consists of the following key components:

1) **Convolutional Layers**: Convolutional layers apply learnable filters (kernels) to the input image, sliding across spatial dimensions (height and width) to produce feature maps. These feature maps encode spatial information such as edges, textures, and shapes, which are crucial for understanding image content. The convolution operation is mathematically expressed as:

$$O(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n) \cdot K(m,n)$$

where I represents the input image, K is the kernel, and O is the output feature map. The kernel's size and stride determine how much the feature map is reduced, while the activation function (e.g., ReLU) introduces non-linearity, allowing the network to model complex patterns.

2) **Pooling Layers**: Pooling layers downsample the spatial dimensions of feature maps while retaining the most significant information. This operation reduces computational complexity, minimizes overfitting, and improves feature invariance to translations. Common pooling methods include:
   - **Max-Pooling**: Retains the maximum value within each pooling window, emphasizing strong activations.
   - **Average-Pooling**: Computes the average value within the pooling window, offering a smoother representation.

   Max-pooling is often preferred for its ability to highlight prominent features while discarding background noise.

3) **Fully Connected Layers**: After the convolutional and pooling layers extract spatial features, fully connected layers (dense layers) process this information to make predictions. These layers interpret the high-level features by learning complex decision boundaries. The final fully connected layer typically outputs class probabilities using a softmax (for multi-class classification) or sigmoid activation function (for binary classification).

TABLE I
SUMMARY OF CNN COMPONENTS AND THEIR ROLES

Component	Role in CNN
Convolutional Layers	Extract spatial features such as edges, textures, and shapes using learnable filters.
Pooling Layers	Downsample feature maps to reduce dimensions and retain significant information.
Fully Connected Layers	Interpret extracted features and map them to class probabilities for predictions.
Activation Functions	Introduce non-linearity to model complex patterns (e.g., ReLU, Sigmoid).
Dropout	Mitigate overfitting by randomly deactivating neurons during training.

C. Hierarchical Feature Learning

CNNs are particularly effective because of their ability to learn hierarchical features:

- **Low-Level Features**: The initial layers capture basic elements such as edges, corners, and textures.
- **Mid-Level Features**: Intermediate layers detect more complex patterns, such as shapes or motifs.
- **High-Level Features**: The deeper layers identify task-specific features, such as objects or postures, which are crucial for classification tasks.

This hierarchical approach enables CNNs to generalize well across various image classification tasks, including nuanced ones like the ”Cat vs. Cat Loaf” problem.

D. Advantages of CNNs

CNNs offer several advantages over traditional machine learning techniques:

- **Automatic Feature Extraction**: Unlike traditional approaches requiring handcrafted features, CNNs learn features directly from data.
- **Parameter Sharing**: Convolutional layers reuse the same kernel weights across the input, significantly reducing the number of trainable parameters compared to fully connected networks.
- **Spatial Invariance**: Pooling layers make CNNs robust to minor translations, rotations, and distortions in the input image.
- **Scalability**: CNNs perform well on large-scale datasets and are highly adaptable to complex tasks like object detection and segmentation.

E. Relevance to This Project

In this study, CNNs are used to distinguish between subtle posture differences in ”Cat” and ”Cat Loaf” images. The convolutional layers extract posture-specific features, such as compactness or body outline, while pooling layers ensure these features are invariant to minor variations in pose or alignment. The fully connected layers interpret these features, producing a probability score for binary classification. By leveraging the hierarchical learning capabilities of CNNs, this project effectively addresses the challenges of posture-based image classification.

III. Dataset Overview

A. Dataset Description

The "Cat vs. Cat Loaf" dataset consists of 646 RGB images, evenly divided into two classes:

- **Cat**: 323 images of cats in traditional poses.
- **Cat Loaf**: 323 images of cats in a loaf posture, characterized by a compact body with paws tucked underneath.

Images were resized to 100×100 pixels to standardize input dimensions for the CNN. This uniformity simplifies the learning process and ensures that the model focuses on feature extraction rather than data variability.

B. Relevance of the Dataset

This dataset presents a unique challenge in computer vision, as the classification task relies on detecting subtle differences in posture rather than entirely different objects. The problem highlights the importance of spatial feature extraction in neural networks and serves as a testbed for evaluating the robustness of CNN architectures in nuanced classification tasks.

IV. Methodology

A. Data Preprocessing

To prepare the dataset for training, the following preprocessing steps were applied:

- **Resizing and Normalization**: All images were resized to 100×100 pixels, and pixel values were scaled to the range $[0, 1]$ to ensure uniform input and faster model convergence.
- **Data Augmentation**: Random transformations, including rotation (up to 30°), width and height shifts (up to 20%), and horizontal flips, were applied. These augmentations increased dataset diversity, reduced overfitting, and improved generalization.

B. Model Architecture

The CNN architecture consists of:

- Three convolutional layers with filter sizes of 32, 64, and 128, each followed by ReLU activation and max-pooling.
- A fully connected layer with 128 neurons and ReLU activation.
- A dropout layer with a rate of 0.5 to prevent overfitting.
- A final dense layer with sigmoid activation for binary classification.

C. Training and Validation

The dataset was split in an 80:20 ratio, with the majority reserved for training and a smaller subset for validation. The model was trained for 10 epochs using the Adam optimizer and binary cross-entropy loss. Batch size was set to 32, balancing training efficiency and memory requirements.

V. Experimentation and Evaluation

A. Evaluation Metrics

The model's performance was assessed using accuracy, precision, recall, and F1-score. These metrics provided a comprehensive understanding of the model's efficacy. A confusion matrix was used to visualize the distribution of predictions across the "Cat" and "Cat Loaf" classes, highlighting strengths and potential areas for improvement.

B. Experimentation Steps

The experimentation process involved:

1) Loading and preprocessing the dataset, including normalization and augmentation.
2) Training the CNN on the augmented dataset with an 80:20 split for training and validation.
3) Hyperparameter tuning to optimize learning rates, batch sizes, and network depth.
4) Monitoring performance across epochs to identify potential overfitting or underfitting.
5) Testing on unseen data to evaluate real-world applicability.

VI. Results and Discussion

After completing the training process, the model achieved a validation accuracy of 66.15% and a validation loss of 0.62. Figures 1 and 2 show the training and validation accuracy and loss curves.

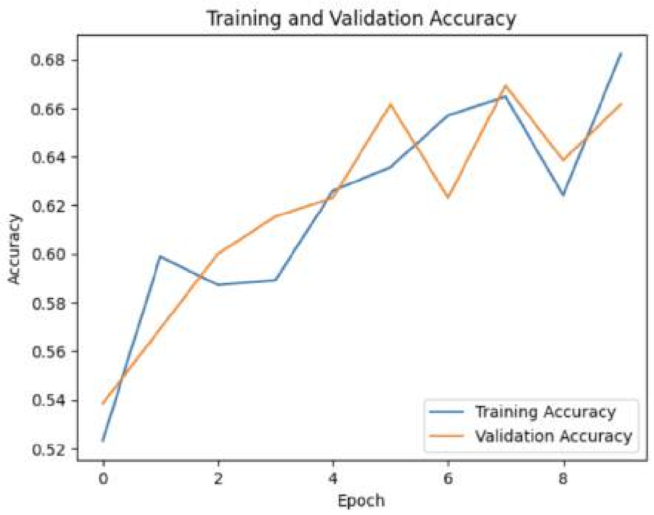


Fig. 1. Training and Validation Accuracy. The accuracy plot illustrates the model's learning progress over the epochs, showing improvement in both training and validation accuracy.

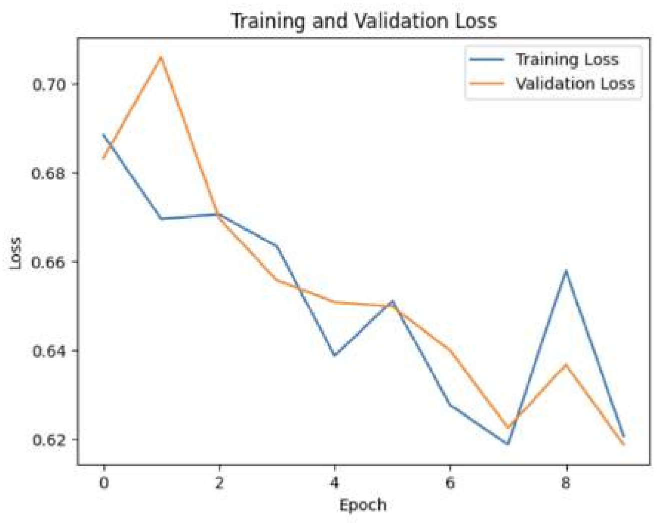


Fig. 2. Training and Validation Loss

The results demonstrate the model's ability to distinguish between the two classes, though subtle posture differences and a limited dataset size present challenges. Future improvements, such as increasing the dataset size or using transfer learning, could enhance performance.

VII. Concept to Code

The implementation of this project, including all scripts for preprocessing, training, evaluation, and visualization, is available in the following GitHub repository:

https://github.com/MHC-FA24-CS341CV/beyond-the-pixels-emerging-computer-vision-research-topics-fa24/blob/main/Image$_classification.ipynb$

This repository contains a Jupyter Notebook detailing the complete pipeline for "Cat vs. Cat Loaf" image classification. The notebook is modular and well-documented, making it accessible for replication and further exploration.

This project's code runs in google collab, and here is code includes:

- **Data Preprocessing**: Scripts for resizing, normalizing, augmenting, and splitting the dataset into training and validation sets.
- **Model Architecture**: A Python script defining the Convolutional Neural Network (CNN) model used for the classification task.
- **Training and Evaluation**: Scripts for training the model, visualizing performance metrics, and saving the trained model.
- **Execution Instructions**: A detailed text file outlining how to run the code step-by-step, including required dependencies and configurations.

A. How to Run the Code

To replicate the results and use the provided code, follow these steps:

1) **Set Up the Environment**:
   - Install Python (version 3.8 or later) and necessary libraries such as TensorFlow, NumPy, Matplotlib, and Pillow.
2) ** add the Dataset from kaggle**:
   - The dataset is available at the repository or via the Kaggle dataset link provided in the documentation.
3) **Run the Preprocessing Script**:
   - Execute the preprocessing script to resize images to 100×100 pixels, normalize pixel values, and split the data into training and validation sets.
4) **Train the Model**:
   - Use the script to train the CNN on the preprocessed dataset.
   - Training progress, including accuracy and loss metrics, will be logged and visualized during execution.
5) **Evaluate the Model**:
   - Evaluate the trained model on validation data using the following command:
   - This will output key performance metrics, such as validation accuracy, precision, recall, and F1-score, as well as a confusion matrix.
6) **Visualize Results**:
   - Visualizations of training and validation accuracy/loss curves can be generated by running the script.
7) **Save and Use the Model**:
   - The trained model is saved in `.h5` format for future inference.
   - Use this model to make predictions on new data by running the script.

B. Next Steps for Technical Implementation

While the current implementation demonstrates reasonable performance, there remain open challenges and areas for improvement in this subfield of posture-based image classification. These challenges include limitations in dataset diversity, feature extraction, and model robustness. The following steps aim to address these limitations and refine the proposed research approach:

- **Expanding the Dataset**: One of the primary challenges in this project is the limited dataset size, which may restrict the model's ability to generalize to unseen images. Adding more images representing diverse cat postures and loaf-like poses is essential. Additionally, sourcing images from varied backgrounds and lighting conditions can enhance the model's ability to handle real-world scenarios.
- **Using Transfer Learning**: Transfer learning, leveraging pre-trained models like VGG16 or ResNet [6], can address the challenge of learning robust features from a small dataset. These models are trained on large datasets like ImageNet and are capable of extracting hierarchical features effectively. Fine-tuning such models on the "Cat

vs. Cat Loaf" dataset could improve classification accuracy and reduce training time.

- **Incorporating Advanced Augmentations**: Data augmentation techniques currently applied (e.g., rotation, flipping, and shifting) could be extended to include more complex transformations such as random occlusions, color jittering, and perspective warping. These advanced augmentations simulate diverse image variations, improving model robustness and mitigating overfitting.
- **Fine-Tuning Hyperparameters**: Optimal hyperparameter tuning is essential for achieving peak model performance. Future steps include systematic exploration of learning rates, batch sizes, number of layers, and kernel sizes using techniques like grid search or Bayesian optimization. This step addresses the limitations of manual tuning by identifying configurations that maximize validation accuracy and minimize loss.
- **Addressing Class Imbalance and Subtlety of Postures**: One open challenge in this domain is the subtlety of differences between "Cat" and "Cat Loaf" classes, which can lead to classification errors. Techniques like focal loss, which focuses on hard-to-classify samples, and class-specific data augmentation can be utilized to overcome these challenges.
- **Exploring Attention Mechanisms**: Incorporating attention mechanisms, such as the attention blocks used in Vision Transformers (ViT) [7], can help the model focus on critical regions of the image, such as the body posture and tucked-in paws. This approach addresses the current limitation of convolutional operations in capturing global context effectively.
- **Research Idea**: Building on the background provided, a unique aspect of the proposed implementation is the combination of hierarchical feature extraction (via CNNs) with attention mechanisms to enhance focus on subtle visual distinctions. By integrating transfer learning, advanced augmentations, and attention modules, this approach aims to improve classification performance beyond conventional CNN methods.

VIII. Conclusion

This study demonstrates the potential of CNNs for nuanced image classification tasks, such as posture recognition. The results underscore the importance of feature extraction and augmentation in achieving robust performance. Future work will focus on increasing dataset size, exploring transfer learning, and employing attention mechanisms to improve classification accuracy.

References

[1] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[2] A. Krizhevsky et al., "ImageNet Classification with Deep Convolutional Neural Networks," *NIPS*, 2012.

[3] K. He et al., "Deep Residual Learning for Image Recognition," *CVPR*, 2016.

[4] R. Szeliski, "Computer Vision: Algorithms and Applications," Springer, 2011.

[5] T. DeVries, G. Taylor, "Dataset Augmentation in Feature Space," *ICLR*, 2017.

[6] K. He et al., "Deep Residual Learning for Image Recognition," *CVPR*, 2016.

[7] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv preprint arXiv:2010.11929*, 2020.

Object Detection

Thao N. Le

Abstract— **You Only Look Once (YOLO) model for object detection has been a popular model being one of the most balanced between detection accuracy and runtime. With older models relying on non-maximum suppression, YOLOv10 [1], the newest YOLO version, improves performance efficiency by introducing NMS-free training. However, in smaller models, YOLOv10 lacks accuracy compared to YOLOv9 by a margin of 1.0%AP to 0.5%AP. This paper aims to propose using YOLOv10 in combination with YOLOv9 to maximize training time by integrating the YOLOv9 model into YOLOv10 at a smaller number of parameters.**

Index Terms— **Object Detection; Computer Vision; Neural Network**

I. Topic Introduction

Object Detection is a core task of Computer Vision, teaching machines to understand the visual world in the way humans do. This is a fast-growing sub-field with applications in medical, car automation, accessibility aid, text recognition, etc.

A. Background

In the early days of object detection, feature extraction used algorithms and image pre-processing to get information from images. Well-known traditional object detection algorithms for example, the Viola-Jones detector (used for face recognition), the Histogram of Oriented Grades detector (efficient object recognition and localization), and the Deformable Parts Model (detect human bodies, automobiles, etc.) bring forth many possibilities to the computer vision field at the time.

Though revolutionary, these methods became time-consuming and inefficient with the invention of Convolutional Neural Network (CNN). As CNN flourished in recent years, two main categories of object detection emerged: one-stage and two-stage object detection. Two-stage detection (R-CNN, Fast R-CNN, etc.) often box the object first and then performs classifications and localization afterward. Since performing two stages is slow, we developed one-stage models like YOLO detecting bounding boxes and classifying objects simultaneously [2, p. 1].

B. Applications

Object detection is used in various situations, one of them is face recognition. Face recognition starts from a simple algorithm involving edge detection, and geometric feature extraction, then evolves to Viola-Jones detector using Haar features and AdaBoost classifiers. Today, using models like Faster Region with Convolutional Neural Network (Faster R-CNN) or Single-Shot Multibox Detector (SSD), the face detection technology has expanded to face recognition, validation, and face tracking [2, p. 14-15]. Another noticeable application of object detection is text detection. With the rise of CNN, text detection went from edge detection-based (using Sobel, Canny, and geometric features) to Deep-learning-based methods (using CNN to extract image features) and improved rapidly over the last 10 years. Although this application is promising, it still faces many challenges with images having different text fonts, complicated backgrounds, multilingual paragraphs, etc [2, p. 16]. And there are many more applications not mentioned here i.e. pedestrians recognition, autonomous vehicles, medical imaging, etc.

II. Existing Methods

In this section, I will summarize two predominant models from two-stage and one-stage strategies in detecting objects. As said above, the two-stage strategy consists of screening for bounding boxes, and then classifying them, which yields high accuracy with the trade of time efficiency. The one-stage strategy uses only one network to detect, therefore, it will run faster but also the accuracy is high overall but not as high as R-CNN.

A. R-CNN Series

1) R-CNN: The Regional Convolutional Neural Network model was proposed in 2014, marking the first time that deep learning was used for Object Detection. It uses selective search to divide the candidate regions on the image, classify them, and regress on the convolutional network to achieve the result. At the time, the model was revolutionizing and achieved 58.5% on the VOC2007 test set. However, their runtime was reported to be inefficient and not suitable for real-time detection.

2) Fast R-CNN: [3] Proposed in 2015, fast R-CNN aims to fix the runtime problem in R-CNN previously. Fast R-CNN uses the spatial pyramid pooling network (SPPNet) to share the burden and merge classification and regression tasks in one network. The model, indeed, was improved in accuracy to 66.9% on VOC2007. However, due to the selective search algorithm, the model is still not suitable for real-life detection.

3) Faster R-CNN: In the same year, faster R-CNN was released replacing the selective search by Regional Proposal Network solving the runtime issue. The accuracy for the model on VOC2007 was improved to 69.9% with a much better run-time. Still, real-life detection is still a challenge due to long training and complex computational processes.

B. *YOLO Series*

1) YOLOv1: Also in 2015, the You Only Look Once model was released. By turning all tasks into one problem for one Neural Network (model from GoogLeNet [4]), the object detection problem was solved in a faster time. The algorithm goes through a grid, predicts bounding boxes, classifies objects, and then, runs them through CNN. Due to the prediction happening concurrently, the localization ability is weak. The model struggles to localize small objects and misses them in the images. Despite that, the model was able to run in real-time and detect objects quickly. [5]

2) YOLOv8: Proposed in 2023, YOLOv8 can perform tasks not only in detection but also in segmentation and classification. Combining the ideas of C3 and Efficient Layer Aggregation to design a new C2f model, which makes an even more lightweight model.

3) YOLOv10: Building on YOLOv8, YOLOv10 looks for improvement in non-maximum suppression. YOLOv10 provides the model with more accuracy and a lighter training weight. Achieving an even higher accuracy with more complex training weight than the model before but for lighter weight, the model is still lagging behind YOLOv9 by 0.5%-1%. [1]

III. Open Challenges

There are still challenges in images with complex backgrounds making it hard for existing models to detect objects. Objects with different poses are also an open problem since current models lack of ability to generalize especially with different shapes. These two problems make video detection a problem since, in the video, the background and object's pose change drastically. Data bias, computational efficiency, and small object detection are some of the other open challenges in the field.

IV. Concept to Code

A. *Method*

To combine YOLOv9 and YOLOv10, I need to understand at what number of parameters YOLOv10 begins to outperform YOLOv9. According to the graph presented by Ao Wang [1, p. 1], at about less than 20, YOLOv10 begins to outperform or at least on par with YOLOv9. The number of Parameters is proportional to the complexity of the model so I tested on the smallest model of both and another close to the smallest model. In Figure 3, we can see that YOLOv10 performs worse with a mean Average Precision (mAP) of only around 60% for the training dataset. But for the close to the smallest model, Figures 1 and 2 show that they are both on par with mAP around 80% for 20 and beyond epochs training.

I proposed a pipeline in which we monitor the CPU/GPU activity and if the RAM usage is above 75%, we will drop the model YOLOv10 down to a lighter-weight model and when reaching the smallest, we will switch to YOLOv9t instead of YOLOv10n.

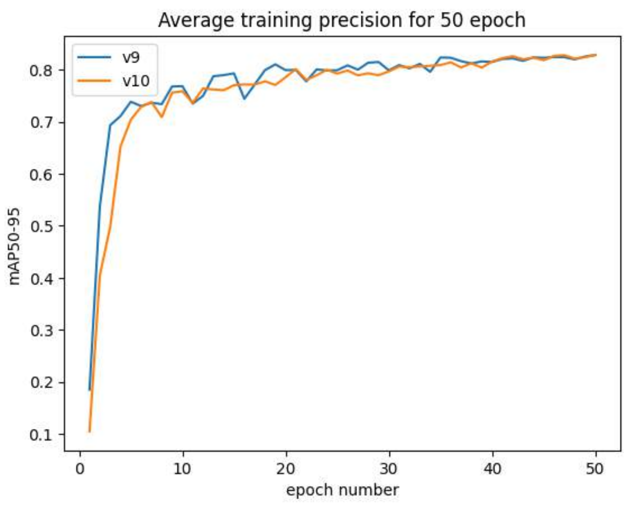


Fig. 1. The graph describes the mean Average Probability (mAP) of Intersection Over Union (IOU) 50-95 over the number of epochs of YOLOv9s and YOLOv10s. We can see that with 50 epochs, the two models provide similar accuracy over all epochs.

Epoch	YOLOv9	YOLOv10
20	0.803 32	0.791 39
50	0.796 13	0.793 62
100	0.819 88	0.826 65

TABLE I

The table show the nearly equal behavior between YOLOv9s and YOLOv10s in different epoch sizes.

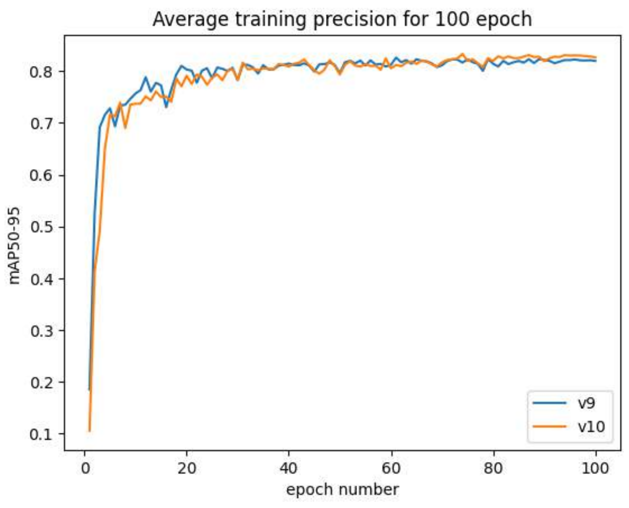


Fig. 2. The graph describes the mean Average Probability (mAP) of Intersection Over Union (IOU) 50-95 over the number of epochs of YOLOv9s and YOLOv10s. We can see that with 100 epochs, the two models provide similar accuracy over all epochs.

B. *Dataset*

The test dataset is the chess piece dataset on roboflow. e

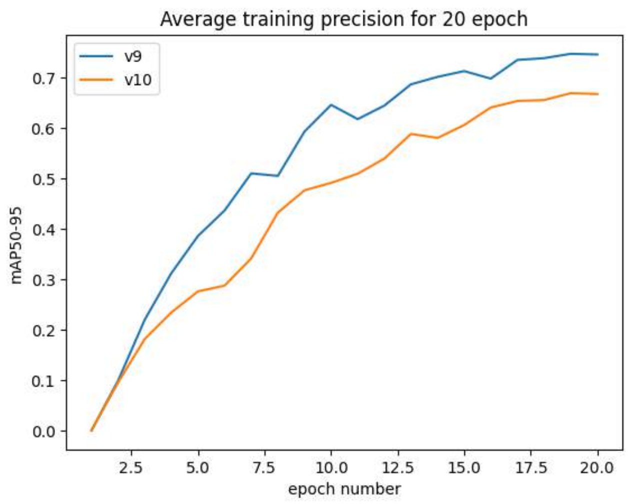


Fig. 3. Epoch20 on yolo9t and 10n

C. *Instruction*

Please see the details in the Python Notebook. The Notebook includes the instructions and also how the graphs were generated.

D. *Future Work*

For future work, I want to successfully have the multiprocess running on Google Colab. With that, I will be able to streamline the idea of changing the model to find the most optimal model for any system to run.

References

[1] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, "YOLOv10: Real-time end-to-end object detection." [Online]. Available: http://arxiv.org/abs/2405.14458

[2] Z. Li, Y. Dong, L. Shen, Y. Liu, Y. Pei, H. Yang, L. Zheng, and J. Ma, "Development and challenges of object detection: A survey," vol. 598, p. 128102. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231224008737

[3] R. Girshick, "Fast r-CNN." [Online]. Available: http://arxiv.org/abs/1504.08083

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions." [Online]. Available: http://arxiv.org/abs/1409.4842

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection." [Online]. Available: http://arxiv.org/abs/1506.02640

Style Transfer in Computer Vision

Miriam Xu

***Abstract*— This paper evaluates and compares the performance of three different approaches to mitigate visual artifacts in style transfer: multi-scale, patch-based, and regularization. Additionally, I offer an overview of the style transfer field, reviewing current methods and analyzing their limitations.**

***Index Terms*— content; style;**

I. Introduction

Style transfer involves modifying the visual style of an image while retaining its content. This process makes images adopt the artistic qualities of another image, allowing for creative and practical applications such as generating art, creating filters, and enhancing visual effects in media and entertainment.

Formally, style transfer algorithms aim to isolate and apply the ”style” features of one image to the ”content” structure of another, achieving an output that combines the two.

A. Concepts

The basic concepts in style transfer rely on the distinction between content and style. The content contains an image's primary structure or layout, usually composed of objects, shapes, and spatial relationships that define the recognizable parts of the scene. Style contains the texture, colors, patterns, and other visual characteristics that contribute to the image's overall aesthetic.

For example, take these two images and their style transfer output:

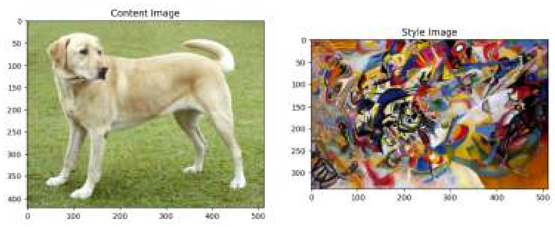


Fig. 1. Example of a content image and style image.

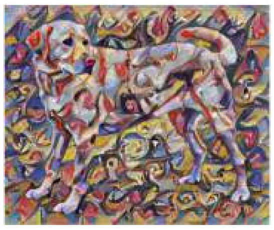


Fig. 2. Example output of style transfer.

The goal of style transfer is to generate an output image I_g that balances the content of a content image I_c with the style of a style image I_s.

This balance can be expressed as an optimization problem:

$$I_g = argminL(I, I_c, I_s)$$

where L is a loss function that measures deviations from both the content and style. The workflow of style transfer can be summarized as:

1) Decompose the input images into content and style components
2) Transform the style features of I_s to match the content structure of I_c
3) Synthesize the new image I_g.

While the specific definition of L varies by method, it typically includes a content term that penalizes deviations from the content of I_c, and a style term that penalizes deviations from the style of I_s. Some methods also incorporate regularization terms to ensure smoothness or naturalness in the generated image.

Style transfer is achieved by iteratively adjusting the pixels of a generated image until the content and style losses are minimized. This optimization process ensures that the generated image converges to one that simultaneously resembles the content image in structure and the style image in visual texture. Continuing the example from the previous figures, here is I_g formed with a shorter optimization:

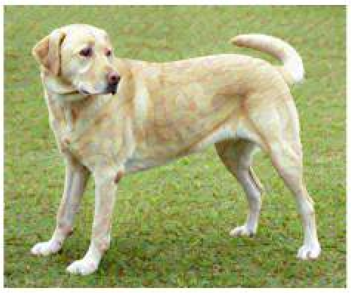


Fig. 3. I_g with shorter optimization.

As you may observe, the content is overpowering the style. The functions used for loss minimization typically include content loss, style loss, perceptual loss, and regularization.

Content loss measures how well the output image I_g retains the structural or semantic feature of the content image I_c. This is often quantified using pixel-level similarity, such as mean squared error, or feature-based similarity.

Style loss measures the stylistic similarity between the output image I_g and the style image I_s. Depending on the method, this may involve comparing textures, color distributions, or statistical properties such as Gram matrices.

Perceptual loss uses human-perceptual metrics from pre-trained neural networks to ensure the output image is coherent.

Regularization in the form of smoothness constraints such as total variation loss can be added to prevent artifacts in I_g.

B. Notation and Syntax

Throughout this paper, we denote:
I_c : Content image.
I_s : Style image.
I_g : Generated image.

II. Existing Methods

Style transfer in computer vision has evolved significantly, mainly leveraging deep learning to blend content and style from images. Early methods were built on neural network-based approaches, such as Gatys et al.'s Neural Style Transfer (NST) [1].

Recent advancements have moved towards real-time style transfer using feedforward networks. Models like Johnson et al.'s Perceptual Loss Networks precompute style information during training, allowing for faster inference. Other works, like Huang and Belongie's Adaptive Instance Normalization (AdaIN), introduced mechanisms for more flexible and diverse style control by aligning content features to style statistics [2].

State-of-the-art methods include GAN-based approaches (e.g., StarGAN and CycleGAN) and diffusion models. GANs focus on generating high-quality and diverse styles while maintaining content fidelity [3].

A. Neural Style Transfer

Neural style transfer uses convolutional neural networks (CNNs) to optimize image representations for combining content and style. These methods calculated content and style losses based on feature activations in a pretrained CNN, such as VGG, and iteratively adjusted pixel values to create stylized outputs. Gram matrices are used to quantify the correlations between feature maps within a CNN layer [1].

Given a layer with N feature maps of size M, the Gram matrix G of a layer l is defined as:

$$G^l_i j = \sum_k F^l_i k F^l_j k$$

where F represents the feature map activations of layer l, and $G^l_i j$ encodes the degree of co-activation between feature maps i and j, capturing textural details indicative of style.

Content loss, $L_{content}$, denotes the difference between the content of the generated image and the content of the original image, typically as the mean squared error between feature maps:

$$L_{content} = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{H} \sum_{k=1}^{W} (F^l_{ijk}(I_g) - F^l_{ijk}(I_c))^2$$

where F^l_{ijk} represents the activation of the i-th feature map at spatial location (j, k).

Style Loss, L_{style}, quantifies the difference in style between the generated image and the style image S by comparing their Gram matrices G^l:

$$L_{style}(S, G) = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{i,j}(S) - G^l_{i,j}(G))^2$$

Where N_l and N_l denote the number of filters and spatial size at layer l.

The total loss function L combines these:

$$L = \alpha L_{content} + \beta L_{style}$$

where α and β are weighting factors that control the emphasis on content and style, respectively.

B. Feedforward Networks

Feedforward approaches like Johnson et al.'s method minimize the perceptual loss during training. The perceptual loss includes feature reconstruction loss and style reconstruction loss. Feature reconstruction loss is defined as $L_{feature}$:

$$L_{feature} = \sum_l ||\phi_l(G) - \phi_l(C)||_2^2$$

where ϕ_l are feature maps from a pretrained network. Style reconstruction loss is defines as L_{style}:

$$L_{style} = \sum_l ||G_{\phi_l}(G) - G_{\phi_l}(S)||_F^2$$

where G_{ϕ_l} is the Gram matrix at layer l [2].

C. Adaptive Instance Normalization (AdaIN)

AdaIN dynamically aligns the mean and variance of content features to those of the style features:

$$AdaIN(F_c, F_s) = \sigma(F_s)(\frac{F_c - \mu(F_c)}{\sigma(F_c)} + \mu(F_s)$$

where μ and σ are respectively the mean and standard deviation [2].

D. Generative Adversarial Networks (GANs)

GANs like CycleGAN use adversarial losses to map between content and style domains [3]. Adversarial Loss is defined as $L_G AN$:

$$L_G AN = \mathbb{E}[logD(Y)] + \mathbb{E}[log(i - D(G(X)))]$$

where D is the discriminator, and G is the generator. Cycle Consistency Loss is defined as L_{cycle}:

$$L_{cycle} = ||G(F(X)) - X||_1$$

III. Open Challenges

A. Current Limitations

Many open challenges are still present in style transfer; finding the right balance between I_s and I_c and creating a model that is both fast and high resolution, achieving high-quality results in real-time while preserving the content structure proves to be difficult. The issue I'd like to address is spatial consistency and artifacts.

Style transfer can introduce unnatural textures, distortions, or other artifacts, particularly when transferring highly detailed or intricate styles. Loss functions that penalize spatial inconsistency, along with patch-based methods that apply style in small, image-appropriate patches, have been explored. Temporal consistency techniques have also been introduced to avoid flickering in video style transfer. However, some artifacts continue to persist, especially around edges or high-contrast areas. Additionally, these methods are computationally intensive and do not always generalize well, especially when there are overlapping or fine details [1].

IV. Proposing Method

This research proposes to evaluate and compare three prominent techniques for reducing artifacts in style transfer: multi-scale style transfer, patch-based approach, and regularization in training.

A. Multi-scale Style Transfer

Multi-scale style transfer works by applying style transfer at multiple scales, progressively refining the output from coarse to fine detail. By stylizing the image at different resolutions and blending the results, the idea is that we can achieve a more cohesive look [1]. The general steps include:

1) Iteratively decrease the scale of the content image I_c to a set of five progressively smaller resolutions.
2) Starting from the lowest resolution, apply the style transfer at each scale.
3) Blend the output incrementally, using the result of the previous scale as a guide for the next.
4) Increase the scale of the final output to the original resolution.

B. Patch-based Approach

The patch-based approach treats the style transfer problem as a local matching task. Instead of applying the style globally across the entire image, they decompose both the content and style images into small patches. The style transfer process matches and applies style patterns patch by patch [4]. The general steps include:

1) Divide both I_c and I_s into small overlapping patches
2) Compare each patch in I_c to patches in I_s, finding the best match by comparing similarities between corresponding pixels.
3) Replace the content patches with the corresponding style patches.
4) Use a blending function to combine overlapping patches smoothly.

C. Regularization

Regularization techniques involve adding constraints to the loss function during the training of the style transfer model. These constraints help smooth the output image, reducing noise and preventing abrupt transitions that cause artifacts [5]. I will be applying total variation loss, which penalizes abrupt intensity changes between adjacent pixels:

$$L_{TV} = \sum_{i,j}((I_{i,j} - I^2_{i+1,j} + (I_{i,j} - I_{i,j+1})^2)$$

where I is the generated image, i is the row index of the pixel, and j is the column index of the pixel.

D. Comparison

Here is a comparison review of the three approaches gained from initial research.

Aspect	**Multi-Scale**	**Patch-Based**	**Regularization**
Artifact Reduction	Global and local patterns	Textures	Noise and Transitions
Content Preservation	High	Moderate	High
Style Consistency	High	High	Moderate
Computational Cost	High	High	Low to Moderate
Ease of Implementation	Moderate	Moderate to Difficult	Easy to Moderate

TABLE I
Comparison of Techniques for Artifact Reduction in Style Transfer

V. Concept to Code

The program will use the datasets to evaluate the performance of each method across varying content and style, and compare outputs based on artifact presence and coherence. I will be building off of this code to implement the three methods. The source code uses neural style transfer using the VGG19 network architecture, a widely-used, pretrained image classification network. The program extracts the style and content, then runs a gradient descent using a Tensorflow Hub pretrained model. To run the code, simply open in CoLab and run all cells.

A. Future Steps

For testing, I will use the MS COCO (Microsoft Common Objects in Context) dataset as content images and INNAT's Van Gogh Paintings dataset for the style images.

Preprocessing must be implemented to:

1) Resize and normalize the images.
2) Extract 16x16 patches from images for patch-based approach.
3) Resize I_c for multi-scale approach.

For the multi-scale method, the style and content image will be resized through preprocessing. These images will then be iteratively passed through the preexisting style transfer model.

For the patch-based method, the style and content patches extracted from preprocessing will be compared to each other, matching the patches with the most pixel-level similarity through Euclidean distance. For two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, the Euclidean distance $d(P_1, P_2)$ is:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Then replace each content patch with its corresponding style patch, using Guassian blending to prevent visible seams. The loss function will be adjusted to measure consistency between adjacent patches.

For regularization, I will extend the loss function to include total variation loss and gradient penalties. Weight adjustments will also be added for balancing regularization terms.

Output will be visualized through side-by-side comparisons of content, style, and output images. Additionally, I_c will be overlaid on I_g to better highlight artifacts.

References

[1] M. B. Leon A. Gatys, Alexander S. Ecker, "Image style transfer using convolutional neural networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 9906, no. 2, pp. 694–711, 2016.

[2] Z. F. J. Y. Y. Y. . M. S. Yongcheng Jing, Yezhou Yang, "Neural style transfer: A review," *ArXiv*, no. 4, 2018.

[3] A. Helwan, "Recent advancements in gavs and style transfer," *Medium*, no. 5, 2023.

[4] T. Q. C. . M. Schmidt, "Fast patch-based style transfer of arbitrary style," *arXiv preprint arXiv:1612.04337*, no. 2, 2016.

[5] . L. F.-F. Justin Johnson, Alexandre Alahi, "Perceptual losses for real-time style transfer and super-resolution," *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 9906, no. 3, pp. 694–711, 2016.

Image Super Resolution using GAN framework

***Abstract*— Image Super Resolution is a rising field of computer vision. It aims to enhance the resolution of an image from low to high. Through the development of technology and deep learning, the application of Image Super Resolution ranges from simple resolution enhancement in phones to analysis of medical data such as MRI. There are various ways to perform Image Super Resolution. This paper focuses on SRGAN and ESRGAN applications.**

I. Topic Introduction

Image Super Resolution is a significant system in computer vison and image processing to improve the visual perception of the poor-quality images. The primary goal is to convert blurred, unclear, low-resolution images into sharp, high-resolution images. This process is often referred to by terms such as interpolation, image scaling, enlargement, and upscaling [1]. There are two main approaches to super-resolution (SR): multi-frame and single-frame based on the input low-resolution (LR) data. Multi-frame SR utilizes multiple images of the same scene with slight sub-pixel shifts to leverage and reconstruct a higher-resolution image or sequence. However, in cases where multiple LR images are unavailable, limited LR data is used to generate the high-resolution (HR) image, an approach known as single-frame SR [2].

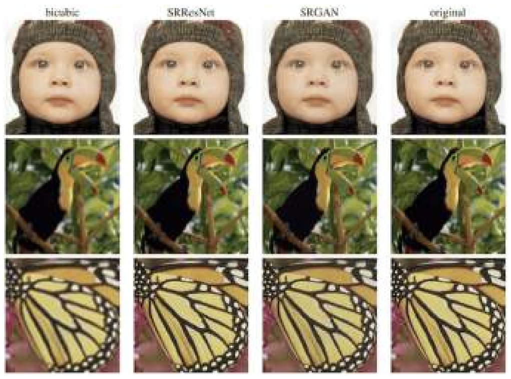


Fig. 1. Examples of Super Image Resolution [3].

II. Existing methods

There are various methods to perform Image Super Resolution as listed in Figure 1. Among all methods, this paper focuses on GAN-based Connections, specifically on SRGAN.

GAN or generative adversarial network is a deep learning model that uses two competing neural networks (generator and discriminator) to generate realistic new data based on a given dataset. The generator creates new data samples,

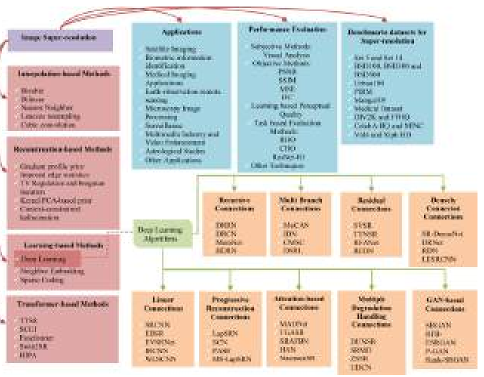


Image Super-resolution
Applications
Performance Evaluation
Benchmark datasets for Super-resolution
Interpolation-based Methods
Reconstruction-based Methods
Learning-based Methods
Deep Learning
Transformer-based Methods
Deep Learning Algorithms
Recursive Connections
Multi Branch Connections
Residual Connections
Densely Connected Connections
Linear Connections
Progressive Reconstruction Connections
Attention-based Connections
Multiple Degradation Handling Connections
GAN-based Connections

Fig. 2. Taxonomy of the existing state-of-the-art super resolution techniques [1].

while the discriminator evaluates if these samples are real or fake compared to the original data. Over time, the generator improves its output until the discriminator can no longer tell the difference between generated and real data [4].

SRGAN (Super-Resolution Generative Adversarial Network) applies a GAN framework to single image super-resolution (SISR). The generator creates high-resolution (HR) images from low-resolution (LR) inputs and a discriminator distinguishes between real and generated HR images. SRGAN enhances the image quality by using a perceptual loss that includes adversarial loss and content loss based on high-level features from a pre-trained VGG network, which is a approach that helps recover realistic image details [5].

ESRGAN (Enhanced SRGAN) improves on SRGAN by enhancing the generator and discriminator. The generator includes Residual-in-Residual Dense Block (RRDB) to increase stability and performance, while the Relativistic GAN discriminator compares how realistic generated images are against real ones, producing sharper and more detailed images. ESRGAN also improves the perceptual loss by using features in the VGG network before activation, improving brightness consistency and texture detail [6].

The evaluation of super-resolution methods is carried out using both objective and subjective assessments to measure image quality. Objective metrics include Peak Signal-to-Noise Ratio (PSNR), which calculates pixel-wise accuracy between super-resolved images and ground truth, and Structural Similarity Index (SSIM), which evaluates structural and perceptual similarities. Subjective evaluation involves techniques like Mean Opinion Score (MOS), where human raters evaluate the visual appeal of images, and qualitative assessments, which examine the sharpness, texture, and absence of artifacts. These methods collectively provide a comprehensive understanding of the effectiveness of super-resolution approaches in generating high-quality, realistic images [5], [6].

The similarities and differences of SRGAN and ESRGAN are summarized in Table 1.

TABLE I
COMPARISON BETWEEN SRGAN AND ESRGAN BASED ON ARCHITECTURAL AND PERFORMANCE IMPROVEMENTS.

Feature	SRGAN	ESRGAN
Network Architecture	Uses standard residual blocks with batch normalization [5].	Introduces Residual-in-Residual Dense Block (RRDB) and removes batch normalization, enhancing stability and feature extraction [6].
Adversarial Loss	Standard GAN discriminator that predicts if an image is real or fake [5].	Uses Relativistic GAN (RaGAN), evaluating the relative realism of generated images against real ones to enhance texture fidelity [6].
Perceptual Loss	Combines adversarial loss with content loss based on VGG network feature maps after activation, which can result in brightness inconsistency [5].	Uses VGG features before activation in perceptual loss, leading to improved brightness consistency and texture detail [6].
Detail and Texture Quality	Improves perceptual quality with photorealistic detail recovery but may still produce artifacts and lack fine detail [5].	Produces sharper and more natural textures, particularly in complex areas such as edges, fur, and grass [6].
Applications	Effective for general single-image super-resolution tasks where moderate visual quality is acceptable [5].	Ideal for applications needing high perceptual quality, such as medical imaging and satellite image enhancement, due to sharper detail recovery [6].

III. OPEN CHALLENGES

A. Current Limitations

Even though GAN model is widely used and well accepted in deep learning field, especially in super image resolution, there are several limitations. Common failures include mode collapse, where the generator maps different inputs to a single output class, resulting in low diversity among samples. Non-convergence can occur when training becomes unstable due to oscillations or divergence between the generator and discriminator. Additionally, vanishing or exploding gradients can slow down or completely halt the learning process entirely [7].

SRGAN challenges occur mostly due to network depth issue because SRGAN are harder to train and they often produce high-frequency artifacts, resulting in repetitive or noisy texture instead of realistic details. Thus, as network depth increases, it becomes harder to balance the adversarial and perceptual loss functions effectively. The deeper layers can exaggerate minor inaccuracies, resulting in repetitive or noisy textures instead of realistic image details. Also, The deeper layers of SRGAN can amplify small inaccuracies, leading to repetitive or noisy textures instead of realistic image details. Additionally, its deep architecture, while improving perceptual quality, requires significant computational resources [5].

ESRGAN often struggles to recover fine local details, leading to blurry or unnatural artifacts. This may be due to the limitations of the pre-trained VGG network, which may not fully capture the potential features in images, thus restricting the generator's capacity to produce highly accurate outputs [8].

Training both SRGAN and ESRGAN is resource-intensive and requires considerable GPU power, making it less practical for real-time applications or deployment on limited-resource devices [5], [6].

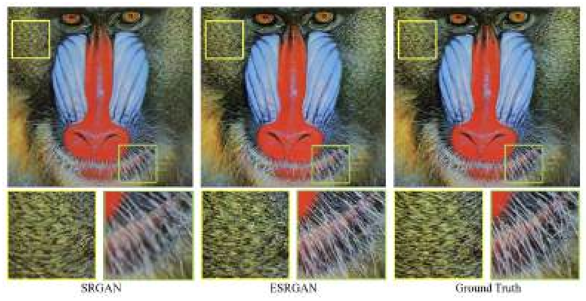


Fig. 3. Image Resolution Comparison between SRGAN, ESRGAN, and ground truth [6]. Ground truth typically represents the original high-resolution images that a model aims to replicate or approximate from low-resolution inputs.

B. Research Idea

This research aims to further develop and apply Image Super Resolution methodologies for enhancing microscopic brain imaging through GAN-Based Super-Resolution. High-resolution imaging of brain structures is crucial for neuroscience research, particularly in mapping complex neural networks and monitoring cellular changes. However, current imaging technologies face physical limitations that hinder the ability to capture ultra-fine details quickly or with minimal photo toxicity. This research seeks to develop a super-resolution model tailored for neuroscience applications, enabling clearer visualization of fine neural structures while reducing the need for ultra-high-magnification imaging. This might be achieved through utilizing techniques that adapt GAN-based models specifically for biological and microscopic imaging contexts such as model training with pre-learned features from high-resolution natural images and gradually fine-tuning it on neuroscience imaging data and incorporating noise reduction techniques or artifact suppression layers into the GAN.

IV. Concept of Code

This code implements the Enhanced Super-Resolution Generative Adversarial Network (ESRGAN) for super-resolving low-resolution images. The objective is to apply ESRGAN to enhance the resolution of specific images (e.g., MRI brain scans), enabling clearer visualization of fine details.

Dataset link: https://www.kaggle.com/api/v1/datasets/download /masoudnickparvar/brain-tumor-mri-dataset

Code link: https://www.kaggle.com/code/nnsssadithya/esrgan-enhanced-super-resolution-gan/comments

1) **Environment Setup and Import Libraries**
2) **Download and Set Paths for Dataset and Model**
3) **Define Constants and Paths**
4) **Image Preprocessing Function**
   - Preprocesses the high-resolution image to make it compatible with the ESRGAN model.
5) **Save and Plot Image**
6) **Load the ESRGAN Model and Generate Super-Resolution Image**
7) **Downscale Image Function (for comparison)**
8) **Display Low-Resolution Image and Calculate PSNR**
   - Calculates the PSNR between the super-resolved and original high-resolution images, providing a metric of quality.
9) **Final Visualization of All Images**
10) **Next Step**
    - As it can be observed, super resolution image is rather lower resolution due to artifact issues as mentioned in open challenges section.
    - The next implementation should be focused on how to improve the resolution when utilizing ESRGAN for medical MRI images.

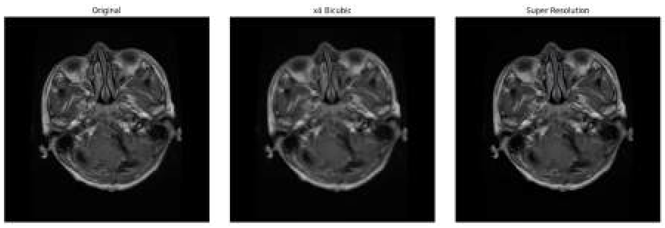


Fig. 4. Output images from sample code: Original, Low resolution, Super resolution.

References

[1] A. D. V. G. Dawa Chyophel Lepcha, Bhawna Goyal, “Image super-resolution: A comprehensive review, recent trends, challenges and applications,” *Infomation Fusion*, vol. 91, pp. 230–260, 2023.

[2] J. L. Q. Y. H. Z. L. Z. Linwei Yue, Huanfeng Shen, “Image super-resolution: The techniques, applications, and future,” *Signla processing*, vol. 128, pp. 389–408, 2016.

[3] J. Brownlee. (2019) 18 impressive applications of generative adversarial networks (gans). Accessed: 2024-11-11. [Online]. Available: https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/

[4] AWS. What is a gan? Accessed: 2024-11-12. [Online]. Available: https://aws.amazon.com/what-is/gan/?nc1=h_ls

[5] F. H. J. C. A. C. A. A. A. A. T. J. T. Z. W. W. S. Christian Ledig, Lucas Theis, “Photo-realistic single image super-resolution using a generative adversarial network,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 105–114, 2017.

[6] S. W. J. G. Y. L. C. D. C. C. L. Y. Q. X. T. Xintao Wang, Ke Yu, “Esrgan: Enhanced super-resolution generative adversarial networks,” *Computer Vision – ECCV 2018 Workshops*, vol. 11133, p. 63–79, 2019.

[7] M. Wiatrak and S. V. Albrecht, “Stabilizing generative adversarial network training: A survey,” *ArXiv*, vol. abs/1910.00927, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID: 203626752

[8] Y. Choi and H. Park, “Improving esrgan with an additional image quality loss,” *Multimed Tools Appl*, vol. 82, p. 3123–3137, 2023.

Face Verification with VGGFace and MTCNN

Robin Tran

Abstract— **The main objective of this project is to enhance the accuracy of face verification, determining if two face images represent the same individual, by combining VGGFace embeddings with Multi-task Cascaded Convolutional Networks (MTCNN) for face detection. Face verification is an important part of computer vision, and is used in various applications, such as identity verification, security, and social media tagging. While VGGFace is widely used for face recognition, it might face challenges in accurately detecting faces in images with complex and noisy backgrounds, which can lead to errors in identity verification. In this project, we use MTCNN as a preprocessing step to accurately detect faces from the images, then apply VGGFace to verify if two images are of the same person. We use the Labeled Faces in the Wild (LFW) dataset for testing, and assess the improvement in verification accuracy by introducing MTCNN.**

Index Terms— **Face recognition; Multi-task Cascaded Convolutional Networks; VGGFace**

I. Introduction

Face verification is an important area within face recognition, where the aim is to confirm whether two face images belong to the same person. In verification tasks, a model generates embeddings, or vector representations, of the two images. If the similarity score between embeddings is above a certain threshold, the faces are considered the same [1].

VGGFace is a deep CNN model designed mainly for face recognition tasks. It was developed by University of Oxford. It produces high-dimensional embeddings that represent unique facial features for similarity comparisons [2].

The **Multi-task Cascaded Convolutional Networks (MTCNN)** model, introduced by Zhang et al., performs face detection and alignment simultaneously [3]. This method is able to detect faces with high accuracy even in complex and clustered environments.

This project aims to explore existing approaches to the current challenge, and specifically, assess whether adding the Multi-task Cascaded Convolutional Networks (MTCNN) as a preprocessing step enhances the accuracy of face verification by VGGFace.

II. Existing Methods and Open Challenges

Several deep learning models have been developed to address the face verification task, with some of the most popular: **FaceNet**, **VGGFace**, and **Siamese Networks** [2] [4]. While these models achieve high accuracy, they might still face challenges, particularly in handling images with cluttered or complex backgrounds, which can affect detection accuracy.

To address these challenges, researchers have experimented with various methods aimed at improving face detection and verification in complex scenes.

A. Skin Color Segmentation

One approach for finding faces in complex backgrounds is skin color segmentation. This method uses color to identify areas in an image that might be skin, then narrowing down potential face regions. By focusing on these areas, the model reduces the search space, which makes locating faces easier [5] However, this approach can be affected by lighting and skin tones. For example, skin color can look different in the natural sunlight and inside light. Although this method is not perfect, skin color segmentation can be a useful first step when combined with other methods to improve face detection.

B. AdaBoost

AdaBoost is an algorithm that combines several simple classifiers to create a stronger one, which can detect faces even in busy backgrounds. It works by selecting the most important features, like edges or textures, that help separate faces from the background. By focusing on only the most useful features, AdaBoost makes face detection faster and more accurate. This method is often used in real-time applications and is especially helpful in images with a lot of background noise, as it narrows down the search area for faces [6]. A limitation of AdaBoost is that it can be sensitive to noisy data or outliers, sometimes leading to false positives. It can also become computationally expensive as it adds classifiers, which may slow down performance in real-time applications [7].

C. Multi-task Cascaded Convolutional Networks

Multi-task Cascaded Convolutional Networks (MTCNN) is a tool that helps with both finding and aligning faces in images, all in one model. It works in three steps. First, it quickly looks for areas that might contain a face. Second, it refines or checks these areas to confirm they're actually faces. Third, it locates key points on the face, like the eyes and nose, to make sure the face is aligned properly. This step-by-step process helps MTCNN accurately detect faces, even in busy or crowded images, and aligns them well [3].

D. Research Idea

Our experiments showed that MTCNN effectively detects and aligns faces, so we want to test if using MTCNN as a preprocessing step can improve VGGFace's accuracy in face verification.

Specifically, we aim to see if MTCNN's role in face detection and alignment, giving VGGFace consistently cropped and centered images, leads to more reliable and accurate embeddings for identity verification. By standardizing the input, we hypothesize that MTCNN preprocessing will produce higher-quality embeddings from VGGFace, improving its ability to verify faces accurately.

To measure the impact of MTCNN preprocessing on VGGFace's accuracy, we will use cosine similarity. This metric checks how similar two embeddings are by calculating the cosine of the angle between them, showing how closely the vector representations match [8].

Cosine similarity, $\cos(\theta)$, is defined as:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{A \cdot B}{||A|| \times ||B||}$$

where A and B are the embeddings of two images, $A \cdot B$ denotes their dot product, and $||A||$ and $||B||$ are the magnitudes of the embeddings. A score close to 1 suggests high similarity (same person), while a score closer to 0 implies lower similarity (different people) [8].

By comparing accuracy of the prediction (based on cosine similarity scores) before and after introducing MTCNN as a preprocessing step, we will assess whether MTCNN preprocessing improves VGGFace's performance in face verification.

III. Concept to Code

In this section, we describe the dataset used and outline the technical steps in the implementation. The code source can be found here https://github.com/MHC-FA24-CS341CV/beyond-the-pixels-emerging-computer-vision-research-topics-fa24/blob/main/code/05-face-recognition/05facerecognition.ipynb.

A. Dataset Description

The **Labeled Faces in the Wild (LFW)** dataset is used to evaluate the face verification accuracy of the proposed approach. LFW is a very popular dataset for face verification tasks, containing over 13,000 labeled images of faces in diverse settings. This dataset challenges the models with variations in background, lighting, and pose, so it is helpful for testing our MTCNN and VGGFace-based pipeline. The dataset is publicly available from the University of Massachusetts Amherst and is downloaded and extracted for testing [9]. We prepare the LFW dataset as image pairs labeled "same" or "different," enabling straightforward comparison of similarity scores for verification.

B. Environment Setup and Run Instructions

To set up and run the code, ensure the following packages are installed in your environment [10]:

- **TensorFlow and Keras**: For building and running the VGGFace model.
- **MTCNN**: For face detection and alignment.
- **NumPy**: For array manipulation and cosine similarity calculations.
- **Matplotlib**: For visualizing face detections and verification results.

We also download the pretrained VGGFace weights from Kaggle, which are used to initialize the VGGFace model and enable it to produce embeddings without having to train again [11].

C. Experiments with MCTNN

We experimented with MCTNN on the LFW dataset, and observed that it can correctly detect and align faces.

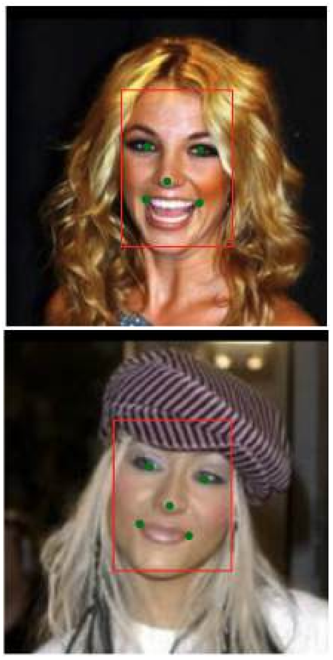


Fig. 1: MTCNN Face Detection and Alignment

D. Pipeline Implementation

The pipeline combines MTCNN for face detection and alignment with VGGFace for embedding generation, aiming to maximize the model's face verification accuracy by providing high-quality inputs.

1) Image Processing:

- **With MTCNN**: Each image is processed through MTCNN, which detects and aligns faces by locating key landmarks (like eyes and nose) and centering each face in the frame. This aligned image is then resized to 224x224 pixels for compatibility with VGGFace.
- **Without MTCNN**: The image is directly resized to 224x224 pixels without any face detection or alignment. This serves as a baseline, allowing us to measure the effect of MTCNN preprocessing.

2) Embeddings: Both MTCNN-preprocessed and baseline images are passed through VGGFace, which generates a high-dimensional embedding for each image. These embeddings represent the face features and are used to determine the similarity between pairs of images [1].

3) Cosine Similarity: For each pair of embeddings, we calculate cosine similarity, which measures how close the embeddings are. A higher cosine similarity score suggests that the two images are likely of the same person, while a lower score suggests different individuals. We define the threshold to be 0.5, so if a pair's cosine similarity is higher than 0.5, the images belong to the same person.

4) Performance Metrics: We evaluate verification accuracy using metrics such as accuracy, precision, recall, and F1 score, comparing VGGFace's performance with and without MTCNN preprocessing. This comparison helps determine if MTCNN preprocessing consistently improves verification accuracy.

E. Model Architecture

The VGGFace model is built to create embeddings that capture the key features of each face. It starts with convolutional layers with filter sizes increasing from 64 to 512, which help pick up on facial details. After each set of convolutions, there are pooling layers that reduce the image size to focus on the most important features and make the model faster. Then, fully connected layers with 4096 filters each combine the features into high-dimensional embeddings. Dropout is used here to help avoid overfitting. Finally, the embedding layer gives a 2622-dimensional vector that represents each face [12].

IV. Output and Results

This section presents the results of the face verification model, comparing the outcomes with and without MTCNN preprocessing.

Firstly, Figure 2 displays example outputs of the model, showing original images alongside their MTCNN-processed counterparts. These figures illustrate how MTCNN preprocessing aligns and centers the faces, which are then fed into the VGGFace model for embedding generation.

We also obtain the performance metrics for face verification with and without MTCNN preprocessing. Table I compares accuracy, precision, recall, and F1 score between the two setups.

Metric	With MTCNN	Without MTCNN
Accuracy	0.558	0.568
Precision	1.000	1.000
Recall	0.116	0.136
F1 Score	0.208	0.239

TABLE I: VGGFace performance with and without MTCNN

The results indicate that, opposite from our hypothesis, adding MTCNN preprocessing had little impact on VGGFace's overall verification accuracy. While MTCNN preprocessing slightly improved precision, making the model more skeptical in identifying matches, it also led to a higher rate of missed same-person pairs. Although the current setup didn't yield significant improvements, we believe that MTCNN preprocessing shows potential and could be explored further to enhance existing deep learning models' performance in addressing difficult face verification tasks.

V. Next Steps

We can compare VGGFace's performance with other state-of-the-art face verification models (such as FaceNet), to assess whether a better solution is possible.

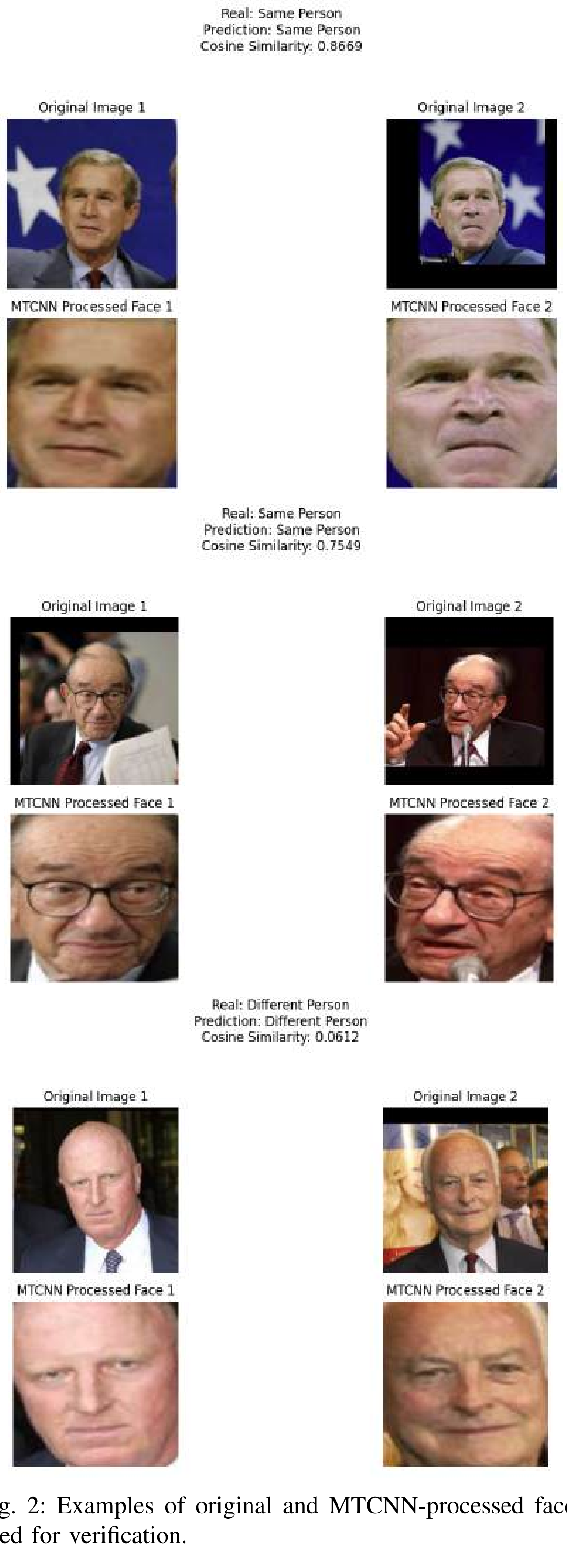


Fig. 2: Examples of original and MTCNN-processed faces used for verification.

References

[1] Y. Huang, J. Xu, and S. Ding, "Face feature embedding," in *Handbook of Face Recognition*, S. Z. Li, A. K. Jain, and J. Deng,

Eds. Springer, Cham, 2024, published 30 December 2023. [Online]. Available: https://doi.org/10.1007/978-3-031-43567-6$_8$

[2] O. M. Parkhi, A. Vedaldi, and A. Zisserman, “Deep face recognition,” in *Proceedings of the British Machine Vision Conference (BMVC)*, 2015.

[3] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, “Joint face detection and alignment using multi-task cascaded convolutional networks,” *IEEE Signal Processing Letters*, 2016.

[4] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” *arXiv preprint arXiv:1503.03832*, 2015, arXiv:1503.03832 [cs.CV]. [Online]. Available: https://doi.org/10.48550/arXiv.1503.03832

[5] Y.-T. Pai, S.-J. Ruan, M.-C. Shie, and Y.-C. Liu, “A simple and accurate color face detection algorithm in complex background,” †E-mail: sjruan@mail.ntust.edu.tw.

[6] P. Viola and M. J. Jones, “Robust real-time face detection,” *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000013087.49260.fb

[7] T. G. Dietterich, “An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization,” *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000. [Online]. Available: https://doi.org/10.1023/A:1007607513941

[8] H. Nguyen and L. Bai, “Cosine similarity metric learning for face verification,” in *Computer Vision – ACCV 2010*, ser. Lecture Notes in Computer Science, R. Kimmel, R. Klette, and A. Sugimoto, Eds., vol. 6493. Springer, Berlin, Heidelberg, 2011, pp. 709–720. [Online]. Available: https://doi.org/10.1007/978-3-642-19309-5$_5$5

[9] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, “Labeled faces in the wild: A database for studying face recognition in unconstrained environments,” University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007. [Online]. Available: http://vis-www.cs.umass.edu/lfw/lfw.pdf

[10] K. N. We utilized several Python libraries for this project, including TensorFlow and Matplotlib.

[11] R. Acharya, “Vgg face model weights for keras models,” updated 4 years ago, available at https://www.kaggle.com.

[12] V. Pillai, “Facial recognition using vggface keras,” updated 2 years ago, available at https://www.kaggle.com.

Cultural Context-Aware Facial Expression Recognition Using Latent Space Clustering

Cynthia Obianuju Akanaga
akana22c@mtholyoke.edu

***Abstract*— Facial Expression Recognition (FER) is an exciting field that aims to come as close as possible to the ability of humans to decipher emotions using simply an image of a person's face. Despite advancements, these systems are often biased due to imbalanced datasets, raising concerns about their fairness. This paper investigates demographic biases in a notable FER dataset, focusing on latent space clustering patterns to reveal disparities in racial groups. Using the UTKFace dataset and ResNet50 model for feature extraction, I applied K-Means clustering to identify patterns in demographic representation. Results show significant underrepresentation of Black individuals, across clusters, underscoring the influence of dataset biases on FER performance. Next steps would be to investigate correlations between other labels (e.g., age, gender) and clustering patterns to uncover multidimensional biases. Also to implement the clustering on other datasets like FairFace and try integrating the underrepresented group into the each cluster to form a balanced set.**

***Index Terms*— Facial Expression Recognition; Latent Space; Feature Extraction**

I. Introduction

Facial expressions may be the universal language of emotion, as theorized by Paul Ekman, a renowned psychologist. Ekman identified seven basic human emotions — happiness, sadness, anger, fear, surprise, disgust, and contempt — that are universally expressed through facial expressions across cultures [1]. However, what happens when the algorithms designed to classify these expressions are unable to accurately read faces for certain demographics? Is it still universal?

Facial Expression Recognition (FER) is a subfield within Affective Computing, the area of artificial intelligence dedicated to understanding and interpreting human emotions to create more personified machines [2]. FER uses patterns in images of people's faces, such as a furrowed brow or a slight pout, to identify emotions like happiness, sadness, anger, or surprise. By analyzing facial expressions to infer emotions, FER offers valuable insights that can be used in diverse fields, from healthcare to autonomous vehicles. However, FER systems are only as good as the datasets they are trained on. The datasets utilized for FER algorithms can be categorized into two types: lab-based and in-the-wild datasets [3].

A. Types of FER Datasets

FER datasets are categorized based on the environment in which the data is collected [3]:

1) Lab-based Datasets: Obtained in controlled environments with consistent lighting, background, and camera angles. Two notable examples are CK+ (Extended Cohn-Kanade dataset and JAFFE dataset). These are unrealistic for real-world applications, as people in everyday scenarios rarely sit up straight, face the camera directly, or pose expressions in ideal lighting conditions.

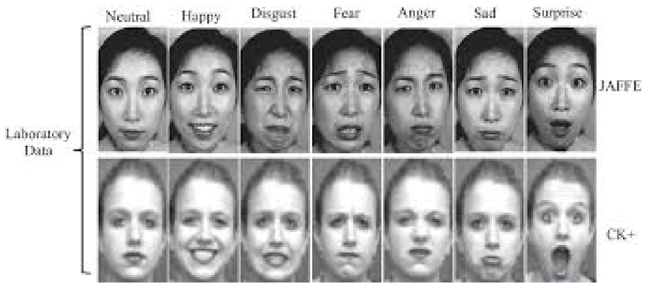


Fig. 1. Example of a Lab-based dataset from JAFFE and CK+ [4], [5].

2) In-the-Wild Datasets: Obtained in natural, uncontrolled environments with diverse lighting, poses, and occlusions (e.g., glasses, masks). An example is the UTKFace dataset, which I used in my running code. These are messier images with challenges like dim lighting, obstructions to the face, and variations in image quality. But it is more realistic and representative of real-world conditions, making it suitable for real-world applications.

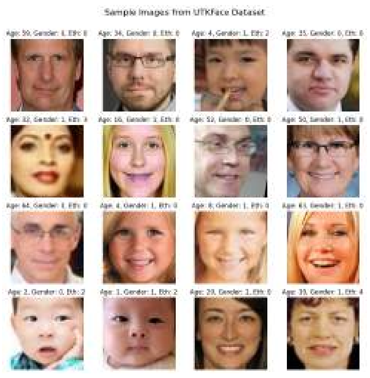


Fig. 2. Example of an in-the-wild dataset from UTKFace [6].

Facial Expression Recognition (FER) systems typically follow three steps: data preprocessing, feature extraction, and classification. The specific methods used for these steps vary depending on whether traditional techniques or deep learning approaches are applied [3].

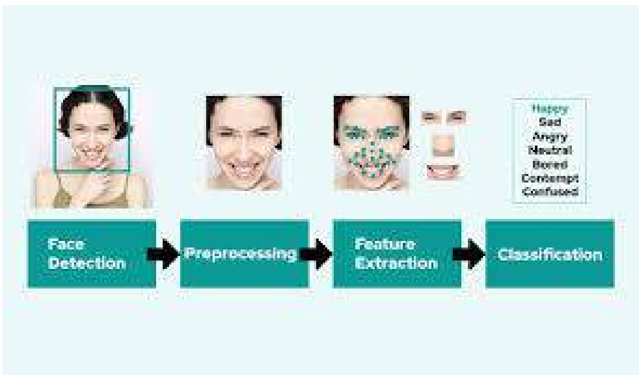


Fig. 3. General FER process [7]

B. Approaches in Facial Expression Recognition

1) Traditional Methods: Traditional methods rely on simple classifiers. The data preprocessing often involves aligning facial images to ensure uniformity, such as detecting landmarks like the eyes, nose, and mouth. Feature extraction uses a descriptor like Local Binary Patterns (LBP) to encode facial expressions into vectors. Finally, classifiers such as Support Vector Machines (SVM) are used to categorize emotions based on these features. [3]

These methods are quick and work well for small datasets, but struggle to generalize to in-the-wild datasets due to their inability to capture complex and diverse patterns in facial expressions.

2) Deep Learning Methods: Deep learning methods automate much of the FER pipeline. The data reprocessing is simplified to basic operations like resizing and normalizing images to prepare them for input into a neural network. Feature extraction is performed automatically by convolutional layers in a deep neural network, such as ResNet50 (I used this for my code) or VGG16, which learn hierarchical representations of facial features directly from data. The extracted features are then passed through fully connected layers to classify emotions. [3]

Deep learning methods outperform traditional techniques in accuracy and generalization, especially on large and diverse datasets. However, with more innovation comes new problems. Since much of the process is automatic, it runs into the problem of making sure the model is learning the right features for expression classification. They are also quite expensive to run and need large amounts of labeled data for effective training. Most of the recent studies in FER utilized teh deep learning method.

C. Key Concepts and Variables for this study

This study employs a clustering-based approach to analyze biases in FER systems. To support this, several key concepts and variables are used:

1) Latent Space: Latent space refers to a high-dimensional feature space where facial images are represented as vectors. These features are extracted from facial images using a deep learning model, such as ResNet50, and capture essential patterns in the data.

2) Clustering with K-Means: Clustering is a method of grouping data points based on their similarity in latent space. This study employs the K-Means algorithm to analyze the latent features of faces. K-Means assigns each image to a cluster, enabling the analysis of how different demographic groups are represented within these clusters.

3) Bias in FER: Bias refers to systematic discrimination in the performance of FER systems across demographic groups. Bias often arises when training datasets are imbalanced or underrepresent certain groups.

II. Existing Methods

In the field of Facial Expression Recognition (FER), both traditional and deep learning methods have faced challenges such as sensitivity to differences in lighting, non-frontal poses, occlusions, and the need for extensive labeled data. Recent research has focused on addressing these limitations to enhance the accuracy and robustness of FER systems.

1) Enhance feature representation: Liao et al. introduced RCL-Net, which combines Local Binary Patterns (LBP) with a residual attention network. This architecture emphasizes local facial details and integrates channel and spatial attention mechanisms to enhance feature representation, improving recognition accuracy in both controlled and wild environments. [3]

2) GAN (Generative Adversarial Networks): Zhang et al. proposed an end-to-end deep learning model based on Generative Adversarial Networks (GANs). Unlike traditional methods that rely on face frontalization or pose-specific classifiers, this model jointly learns to synthesize facial images and perform pose-invariant FER. This is particularly relevant to bias in FER, as it reduces the risk of performance disparities across demographic groups by enriching the training set with more diverse examples. [8]

3) Transformer-based models: Li et al. introduce **FER-former**, a Transformer-based approach designed to advance FER in uncontrolled environments. The method leverages a hybrid stem to combine the strengths of CNNs and Transformers, enabling simultaneous use of image-based features and text-oriented tokens for classification. To address annotation ambiguity, FER-former supervises the similarity between image and text features, aligning image semantics with text-space representations. [9]

III. Open Challenges

A. Current Challenges

Demographic bias remains a significant challenge in Facial Expression Recognition (FER), as many systems fail to account for cultural and individual variations in emotional expressions. While foundational works, such as those by Ekman, have posited the universality of emotions, recent studies highlight the limitations of this assumption when applied to diverse populations [1]. Addressing these concerns, [10] undertook a systematic investigation of bias and fairness in FER systems. They compare baseline, attribute-aware, and disentangled approaches using well-known datasets (RAF-DB and CelebA) and find that the disentangled approach, particularly when combined with data augmentation, is the most effective for mitigating demographic bias. Their findings underscore the importance of designing models that prioritize fairness alongside accuracy, especially in the context of uneven attribute distributions or imbalanced subgroup data.

Despite progress in bias mitigation, open challenges remain. Most facial expression datasets, such as RAF-DB, are skewed toward certain demographic groups, predominantly Caucasian individuals within a narrow age range. Additionally, as noted in [10], larger datasets like CelebA may not benefit significantly from existing augmentation techniques, pointing to the need for alternative approaches to mitigate bias in well-represented data. Many existing approaches, while effective in controlled settings, fall short for in-the-wild datasets.

B. Research Ideas

Given the challenges in mitigating bias in Facial Expression Recognition (FER), my research focuses on leveraging latent space analysis for unsupervised bias detection using the UTKFace dataset. My research aims to identify demographic biases in the model's latent feature space. By analyzing cluster distributions and measuring overlaps, the goal is to reveal subtle biases that may not be captured through conventional fairness metrics.

IV. Concept to Code

A. Dataset

The dataset used for this study is the **UTKFace dataset**, sourced from a publicly available repository. It contains facial images annotated with the following labels:

- **Age:** Numerical values representing the age of the individual.
- **Gender:** Binary values (`0` for male, `1` for female).
- **Ethnicity:** Categorical values corresponding to racial categories (`0` = White, `1` = Black, `2` = Asian, `3` = Indian, `4` = Other, including Hispanic, Latino, and Middle Eastern).

Key Characteristics:

- **Diversity:** The dataset spans a wide range of ages (0–116 years), genders, and racial categories. However, the representation of racial groups is imbalanced:
  - White individuals dominate the dataset (**54.2%**).
  - Black individuals are severely underrepresented (**2.6%**).
- **Preprocessing:** Images are provided in cropped and aligned formats, ensuring that facial features are centralized for analysis.
- **Size:** The dataset contains over 20,000 images. For computational feasibility, a subset of 500 images was used in this project.

The FairFace dataset would have been more ideal, but I decided to use UKFace instead because I struggled to use a dataset that had the labels in a different folder. In the UTKFace dataset, the images are named according to their race, gender and age so I don't need to download anything else, just the images so it was easier and faster to load into Colab.

B. Running the Code

The code can be run by going to the following GitHub folder: `https://github.com/MHC-FA24-CS341CV/beyond-the-pixels-emerging-computer-vision-research-topics-fa24/blob/main/code/06-facial-expression-recognition/06_facial_expression_recognition.ipynb` and downloading the file. Then upload it into google colab and select "Run All" from the Runtime dropdown. Then you can scroll through and view the images and conclusions from the kmeans clustering

C. Results

Using the K-Means algorithm with $n = 4$ clusters, the facial features extracted from the UTKFace dataset were grouped into distinct clusters. Each cluster represents a grouping of similar facial features based on the latent feature space generated by the ResNet50 model. The clusters were analyzed to uncover patterns in racial distribution and potential biases. Key observations:

TABLE I
Ethnicity Distribution Across Clusters

Cluster	White	Black	Asian	Indian	Other
0	58.8%	3.0%	14.1%	13.1%	11.1%
1	51.0%	3.0%	20.0%	14.0%	12.0%
2	36.8%	0.0%	36.8%	21.0%	5.3%
3	52.7%	2.2%	15.9%	18.6%	10.4%

Cluster 0: This was the largest cluster, containing 58.8% White individuals, 14.1% Asian individuals, 13.1% Indian individuals, and smaller proportions of Other (11.1%) and Black (3.0%) individuals. The dominance of White individuals in this cluster reflects the overall dataset bias toward this group.

Cluster 1: This cluster had a more balanced composition compared to others, with 51.0% White individuals, 20.0% Asian individuals, 14.0% Indian individuals, and 12.0% Other individuals. Black individuals were again underrepresented at 3.0%.

Cluster 2: This was the smallest cluster, containing 36.8% White individuals, 36.8% Asian individuals, and 21.0% Indian individuals. Black individuals were absent from this cluster, and Other individuals made up a small proportion (5.3%). The low representation of White individuals in this cluster suggests it captured unique or outlier features not common in the dataset.

Cluster 3: This cluster had 52.7% White individuals, 18.6% Indian individuals, 15.9% Asian individuals, and 10.4% Other individuals. Black individuals accounted for only 2.2%, continuing the trend of underrepresentation.

Cluster 4: Similar to other clusters, White individuals dominated at over 50%, with smaller proportions of Asian, Indian, and Other individuals. Black representation was consistently low.

D. Conclusions

1. Dataset Bias: The clustering results highlight significant imbalances in the dataset. White individuals dominate all clusters, reflecting their overrepresentation in the UTK-Face dataset (54.2%). Black individuals, on the other hand, are severely underrepresented across all clusters, making up only 2.6% of the dataset.

2. Model Bias: The clustering patterns reveal that the latent feature space learned by ResNet50 is influenced by dataset biases. The dominance of White individuals in most clusters suggests the model prioritizes features common to this group, at the expense of underrepresented groups like Black individuals.

V. Future Work

Dataset Balancing: Augment the dataset with additional images of Black individuals and other underrepresented groups to ensure equitable representation.

References

[1] P. Ekman, *Unmasking the Face: A Guide to Recognizing Emotions From Facial Expressions*. Los Altos, CA: Malor Books, 1975.

[2] D. Team. (2022) What is affective computing? Accessed: 2024-11-23. [Online]. Available: https://www.datacamp.com/blog/what-is-affective-computing

[3] J. Liao, Y. Lin, T. Ma, S. He, X. Liu, and G. He, "Facial expression recognition methods in the wild based on fusion feature of attention mechanism and lbp," *Sensors*, vol. 23, no. 9, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/9/4204

[4] M. Lyons, S. Akamatsu, M. Kamachi, and J. Gyoba, "The japanese female facial expression (jaffe) database," 1998, accessed: 2024-11-23. [Online]. Available: https://zenodo.org/record/3451524

[5] T. Kanade, J. F. Cohn, and Y. Tian, "Comprehensive database for facial expression analysis," in *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00)*, 2000, accessed: 2024-11-23.

[6] Z. Zhang, "Utkface: A large-scale dataset of faces with age, gender, and ethnicity labels," 2017, accessed: 2024-11-23. [Online]. Available: https://www.kaggle.com/datasets/jangedoo/utkface-new

[7] A. Biometrics, "How does facial emotion recognition express your feelings?" 2022, accessed: 2024-11-23. [Online]. Available: https://www.aratek.co/news/how-does-facial-emotion-recognition-express-your-feelings

[8] F. Zhang, T. Zhang, Q. Mao, and C. Xu, "Joint pose and expression modeling for facial expression recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3359–3368.

[9] Y. Li, M. Wang, M. Gong, Y. Lu, and L. Liu, "Fer-former: Multi-modal transformer for facial expression recognition," 2023. [Online]. Available: https://arxiv.org/abs/2303.12997

[10] T. Xu, J. White, S. Kalkan, and H. Gunes, "Investigating bias and fairness in facial expression recognition," in *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*. Springer, 2020, pp. 506–523.

Comparative Analysis of Conditional and Super-Resolution Generative Adversarial Networks (CGANs vs SRGANs) for Enhancing Image Resolution

Anh Pham

***Abstract*— This paper investigates the use of Generative Adversarial Networks (GANs) for image super-resolution, focusing on a comparative analysis of Conditional GANs and Super-Resolution GANs. Specifically, this study examines the efficiency of these models in terms of inference speed when enhancing low-resolution images to high-resolution outputs. By conditioning the GAN on low-resolution inputs, CGANs are adapted for super-resolution, allowing a side-by-side comparison with SRGANs, which are specifically designed for this task. This work aims to evaluate the computational trade-offs between these models to inform their application in real-time settings.**

***Index Terms*— image super-resolution; interference time; generator; discriminator**

I. Topic Introduction

Generative Adversarial Networks (GANs) are a type of deep learning architecture used to train a generative model, where the objective is to generate new data that resembles the training data. GANs generate new, realistic data samples that resemble a given dataset, making them powerful for image synthesis, image-to-image translation, and super-resolution tasks in computer vision. These applications make GANs valuable tools in fields like entertainment, healthcare, autonomous driving, and more.

A GAN architecture model involves two sub-models: a generator model for generating new examples and a discriminator model for classifying whether generated examples are real or fake.

- Generator model: This model generates new data samples by learning to map random noise to the data distribution of the target dataset. In image generation, it takes random noise (often a vector of random values) and transforms it into an image.
- Discriminator model: This model acts as a classifier, distinguishing between real images from the training dataset and fake images created by the generator.

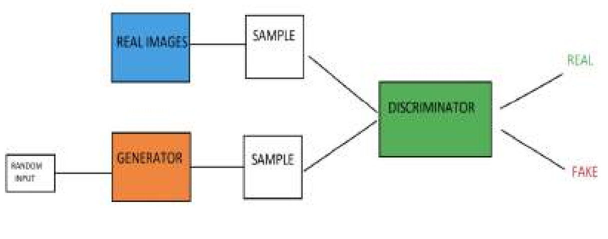


Fig. 1. A high-level diagram of the GAN Generator-Discriminator Infrastructure

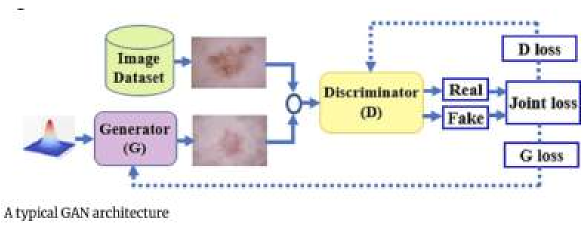


Fig. 2. A typical GAN Generator-Discriminator Infrastructure with its loss functions [1]

In a Generative Adversarial Network (GAN), the underlying mechanism involves the generator striving to produce data that can deceive the discriminator, while the discriminator attempts to differentiate between real and generated data. This adversarial training process fosters continuous improvement in both the generator and discriminator, making them two competitive networks that force each other to be better. This mechanism is reinforced upon examining the GAN loss function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Fig. 3. Standard GAN loss function (min-max GAN loss)

The foundation of GAN is a min-max game in which:

- **Generator**: Minimizes $\log(1 - D(G(z)))$ or equivalently maximizes $-\log D(G(z))$, aiming to fool the discriminator into classifying generated data as real.
- **Discriminator**: Maximizes $\log D(x)$ for real data and $\log(1 - D(G(z)))$ for generated data to distinguish real from fake.

This adversarial interaction drives iterative improvement.

II. Existing Methods

GANs have been widely adopted for image super-resolution due to their ability to generate high-quality outputs. Conditional GANs (CGANs) extend traditional GANs by incorporating auxiliary conditions, such as image-specific features or labels, to guide the generation process. These conditions help CGANs achieve more tailored and accurate outputs. On the other hand, Super Resolution GANs (SRGANs) introduce perceptual loss functions, combining pixel-wise and feature-space losses, to produce visually realistic super-resolved images. While effective, SRGAN's training process remains challenging due to potential instability between the generator and discriminator networks. Both SRGAN and CGAN provide distinct advantages for super-resolution tasks, but their inference speed and performance have not been directly compared in practical applications. This paper investigates these models to evaluate their efficiency and suitability for real-world super-resolution applications.

A. *Conditional GANs*

Conditional GANs (CGANs) are an extension of this standard GAN model that incorporate additional information, or conditions, into both the generator and discriminator networks. They can be used for a wide range of purposes such as conditioning on labels to generate specific classes of images (e.g. dogs, cats).

In a CGAN, the generator takes both random noise and the condition as inputs. This condition can be any extra information that provides specifications as to what the generated output should look like, such as a class label, a feature vector, or another low-resolution version of an image. The discriminator also receives both the generated (or real) data and the condition, allowing it to evaluate whether the generated data aligns with the given condition. This setup helps the discriminator learn to recognize whether the generator's output is both realistic and correctly conditioned.

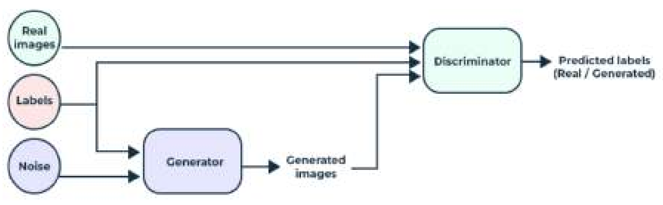


Fig. 4. CGAN Generator-Discriminator Infrastructure

The loss function for a Conditional GAN extends the standard GAN. It fundamentally plays the same min-max game but using conditional probability for both the generator and the discriminator. [2] [3]

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x|y)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z|y)|y))].$$

B. *CGANs for Image Super-Resolution*

While CGANs are typically not used for image super-resolution, Qiao et al. (2019) suggests a CGAN model that can be used in a highly efficient manner for super-resolution purposes. This research presents using a type of CGAN specifically for super-resolution called SRCGAN or Super-Resolution Conditional GAN. By conditioning the discriminator on ground-truth images, SRCGAN inputes a low-resolution image to the generator and upscales it to a high-resolution version. The generator uses a deep residual network with convolutional layers and learns to upscale low-resolution (LR) images to high-resolution (HR) outputs while the discriminator distinguishes real from generated images. This mechanism provides the generator with feedback to improve image resolution quality. [4]

C. *Super-Resolution GANs*

Super-Resolution GANs (SRGANs) are a subtype of Generative Adversarial Networks that are specifically built for image super-resolution ie. increasing the resolution of a low-resolution image to create a higher-quality, more detailed version. They achieve this goal by having the generator model generate finer details and textures of the inputted image. [5]

SRGANs use a GAN framework tailored to generate realistic high-resolution images from low-resolution inputs, specifically trained on image quality. The generator takes in a low-resolution image as input and attempts to generate a high-resolution output by learning patterns and fine details from the data. It uses techniques like residual blocks and upsampling layers to enhance the resolution and add realistic texture to the output. The discriminator evaluates the generated high-resolution image and tries to distinguish it from real high-resolution images. It provides feedback to the generator about how realistic its output is, encouraging the generator to produce more convincing images.

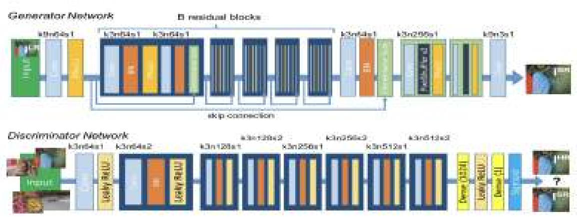


Fig. 5. GAN Generator-Discriminator Infrastructure [5]

The SRGAN loss function distinguishes itself from standard GAN and CGAN by incorporating content loss, derived from feature comparisons in a pre-trained VGG (Visual Geometry Group) network, alongside the traditional adversarial loss. This perceptual loss ensures that the generated image not only fools the discriminator (adversarial goal) but also closely matches the high-resolution image in terms of perceptually relevant features, rather than just pixel-level similarity. This combination enables SRGAN to achieve visually superior super-resolution results. The description and mathematical principles of SRGAN is beyond the scope and the intended length of this short research paper, thus more details can be found in the source cited. [6]

III. Current Limitations

A. *Existing Studies on GAN Performance Comparisons*

Based on a review of existing literature via Google Scholar, it appears that the body of research focused on performance comparisons among GANs remains relatively limited. One such study by B. S. et al. (2023) found that the performance of various GAN architectures — such as DCGAN, SRGAN, and CGAN — varies significantly depending on the dataset and architecture used, with no single model consistently outperforming others across all tasks [7]. In Jozdani et al. (2022), SRGAN and CGAN are compared for image generation in remote sensing. This comparison suggested that SRGANs excels in tasks like super-resolution, while CGAN is more suitable for scenarios requiring controlled image outputs through conditional inputs [8]. The results of previous research are summarized in the table below.

TABLE I
SUMMARY OF COMPARATIVE STUDIES ON GANS

Study	**Findings**
B. S. et al. (2023) *A Comparative Analysis on the Effectiveness of GAN Performance* [7]	Evaluated GANs such as DC-GAN, SRGAN, and CGAN. Found performance depends on dataset and architecture; no single model excels across all tasks.
Jozdani et al. (2022) *A Review and Meta-Analysis of Generative Adversarial Networks and Their Applications in Remote Sensing* [8]	Compared SRGAN and CGAN for remote sensing. SRGAN excels in super-resolution, while CGAN is better for controlled image generation using conditional inputs.

In the subfield of Generative Adversarial Networks (GANs) for image generation, several key limitations persist, particularly in balancing image quality and computational efficiency. These trade-offs are important for future research including mine. Models like SRGAN excel in high-resolution image generation but are computationally demanding, making them impractical for real-time applications. CGANs provide greater control through conditioning on additional inputs but often struggle with image quality when the conditional information is sparse or poorly defined.

Another significant challenge lies in generalization across different datasets. GANs, including SRGAN and CGAN, are highly dependent on the dataset they are trained on. Thus performance can degrade when applied to new, unseen data or in domains where labeled data is limited.

IV. RESEARCH IDEAS

The present paper evaluates the computational trade-offs between CGANs and SRGANs for enhancing image resolution. Using the MNIST dataset, the study compares the efficiency and image quality of both models, with SRGAN modified to handle low-to-high-resolution transformations. The goal is to assess the balance between computational efficiency and image quality. By investigating these models on a well-established dataset like MNIST, the research provides insights that can inform the selection of GAN architectures in real-world applications under resource constraints.

V. RESEARCH METHODS

To achieve the goal, I will leverage TensorFlow and NumPy for easy loading and manipulation of the MNIST dataset, while implementing both CGAN and SRGAN models based on existing GitHub code. Techniques such as perceptual loss for SRGANs will be incorporated to assess how well these models balance high-resolution output with computational load.

For comparisons to be made, the objects of such comparison need to be comparable. The proposed method involves modifying CGANs into SRGANs to compare their capabilities in generating high-resolution images from low-resolution inputs. This approach is sensible because both models share a similar underlying architecture, consisting of a generator and a discriminator.

- **Architecture**: Both CGAN and SRGAN share the same generator-discriminator structure. While CGAN generates images from random noise and class labels, SRGAN generates high-resolution images from low-resolution inputs.
- **Scale**: Both models are applied to the MNIST dataset images (28x28 pixels), allowing a fair comparison of computational efficiency, network complexity, and training time.
- **Discriminator**: Both models use a similar discriminator to distinguish between real and generated images, evaluating either digit outputs (CGAN) or enhanced high-resolution images (SRGAN), enabling consistent comparison.

VI. CONCEPT TO CODE

GitHub source code folder link:
GAN.ipynb [9]

A. Dataset Description

This study uses the Modified National Institute of Standards and Technology (MNIST) dataset. It is a widely recognized benchmark for machine learning and computer vision tasks. The details are as follows:

- **Content:** 70,000 grayscale images of handwritten digits, ranging from 0 to 9.
- **Image Size:** Each image has dimensions of 28×28 pixels.
- **Purpose:** Primarily used for training machine learning models, particularly for digit classification tasks.
- **Data Split:** The dataset is divided into 60,000 images for training and 10,000 images for testing.
- **File Format:** Stored in the `.npz` format (NumPy compressed file), making it easily accessible in Python through libraries like TensorFlow or NumPy.

B. Run Instructions

Run the first code block for the CGANs implementation. Then run the second code block for the SRGANs implementation. Compare the time output and observe the quality of the image generated. Assess the trade-offs.

C. Connection to Proposed Research Idea

This code compares two GANs implementation - CGANs and SRGANs on the same dataset and demonstrated the time trade-offs when using the two models. The code has an implementation of SRGANs based on the upscaling mechanism from CGANs to SRGANs. The perceptual loss function implementation was unsuccessful due to mismatch between expected input and the actual output of the function. Further understanding of mathematical and statistical principles are needed to correctly implement the loss function to replicate a comparable version of SRGANs from our original model CGANs.

D. Technical Implementations for the Next Step

To compare the performance of the CGAN and SRGAN models, particularly in terms of training time, a few strategies in the next step are recommended to measure and track performance efficiently. The code is partially implemented and cannot yet to be run until the SRGANs implementation is successful.

1) Time Tracking with Python's `time` Module
   - *Start and End Time:* Track the start and end times for each training loop to compare the total time taken for training each model.
   - *Epoch-wise Comparison:* Track the time per epoch for both CGAN and SRGAN to see how long each epoch takes for training.
   - *Functionality:* This allows us to isolate where the training time is spent (e.g., generator vs. discriminator updates, image generation, etc.).

VII. Results & Discussion

The code implementation, if successfully run in the future, will demonstrate that one of the two models performed better on the specific dataset MNIST. It is important to consider the nature and characteristic of the MNIST dataset and how they play into the experimental result. MNIST dataset is widely used for training machine learning models, particularly for digit classification, but it may not be the best choice for evaluating image resolution enhancement that may demand high-quality, complex image outputs. MNIST consists of 70,000 grayscale images of handwritten digits (28x28 pixels), which are relatively simple and low-resolution compared to the high-resolution image generation tasks often associated with real-time applications in fields like computer vision or remote sensing. However, even if MNIST was a relatively reasonable choice of dataset within our runtime and computational constraints, the stability of CGANs and SRGANs models have to be tested across diverse datasets for a direct conclusion from our demonstration to be reliable. Based on previous literature, it is plausible that our experiment would largely be inconclusive if not given all comprehensive factors such as the scale and nature of datasets and the specific infrastructure used to run GANs.

VIII. Additional Project Materials

- Presentation Deck: GANs - Google Slide Presentation

References

[1] E. Goceri, "Gan based augmentation using a hybrid loss function for dermoscopy images," *Artif Intell Rev 57*, vol. 234, 2024.
[2] H. Dwivedi, "Understanding gan loss functions," *neptune.ai*, 2023. [Online]. Available: https://neptune.ai/blog/gan-loss-functions
[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.
[4] J. Qiao, "Image super-resolution using conditional generative adversarial network," *IET Image Processing*, vol. 13, pp. 2673–2679, 2019.
[5] C. Ledig, "Photo-realistic single image super-resolution using a generative adversarial network," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 105–114, 2017.
[6] D. Chakraborty, "Super-resolution generative adversarial networks (srgan)," 2022. [Online]. Available: https://pyimagesearch.com/2022/06/06/super-resolution-generative-adversarial-networks-srgan/
[7] C. BS, M. Shreyas, I. Karanth, B. S, A. R. KP, and G. S, "A comparative analysis on the effectiveness of gan performance," pp. 1–8, 2023. [Online]. Available: 10.1109/ICCCNT56998.2023.10307295
[8] S. Jozdani, D. Chen, D. Pouliot, and B. Alan Johnson, "A review and meta-analysis of generative adversarial networks and their applications in remote sensing," *International Journal of Applied Earth Observation and Geoinformation*, vol. 108, 2022. [Online]. Available: https://doi.org/10.1016/j.jag.2022.102734
[9] GeeksforGeeks, "Conditional gans (cgans) for image generation," 2024. [Online]. Available: https://www.geeksforgeeks.org/conditional-gans-cgans-for-image-generation/

Enhancing Image Inpainting with GLIDE: Techniques, Applications, and Performance Evaluation

Abstract— **Image inpainting is a crucial field in computer vision and image processing that has garnered significant attention in recent years. Advances in deep learning have propelled inpainting techniques to new levels of sophistication, with models like GLIDE (Guided Language-to-Image Diffusion for Generation and Editing) emerging as powerful tools for reconstructing missing or damaged regions in images. This paper provides an introduction to GLIDE-based inpainting methods and evaluates their performance in reconstructing images.**

I. Introduction to image inpainting

Have you ever wondered how damaged photographs are restored to look like new, or how satellite images fill in missing data from incomplete scans? The answer is through image inpainting, a technique for filling in missing or corrupted parts of an image by using information from surrounding areas so they blend seamlessly with the rest of the image [1]. With the development of advanced image processing techniques, image inpainting has become widely applicable across fields such as photography, medical imaging, and satellite technology. [2].

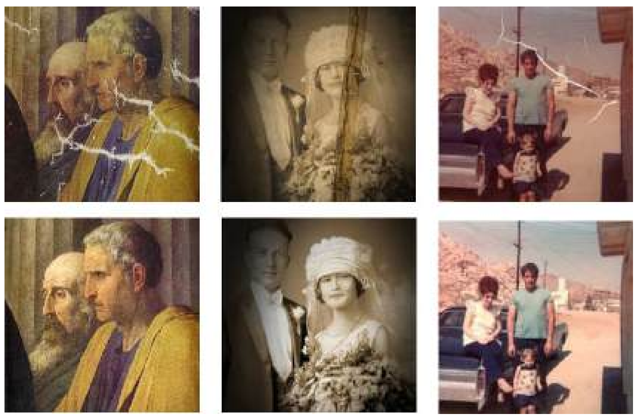


Fig. 1. Historical image inpainting examples, first row presents the original images, second row presents the inpainted images. [3].

II. Introduction to GLIDE

GLIDE (Guided Language-to-Image Diffusion for Generation and Editing) is a powerful AI model developed by OpenAI for photorealistic image generation and editing tasks [4]. The model consists of three main components: a diffusion model (ADM-G) for generating 64x64 pixel images, a transformer-based text encoder, and an upsampling model for 256x256 pixel outputs [5].

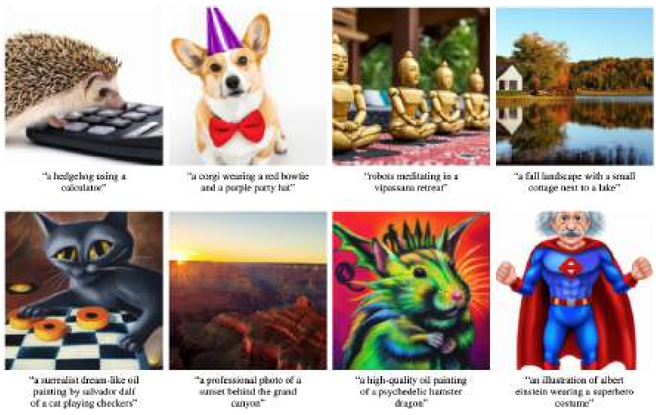

"a hedgehog using a calculator"
"a corgi wearing a red bowtie and a purple party hat"
"robots meditating in a vipassana retreat"
"a fall landscape with a small cottage next to a lake"
"a surrealist dream-like oil painting by salvador dalí of a cat playing checkers"
"a professional photo of a sunset behind the grand canyon"
"a high-quality oil painting of a psychedelic hamster dragon"
"an illustration of albert einstein wearing a superhero costume"

Fig. 2. Examples of images generated by GLIDE based on text prompts. Source: [4].
.

III. Applications of GLIDE in Image Inpainting

A. Effectiveness in Inpainting Tasks

One of GLIDE's unique capabilities is its effectiveness in image inpainting. By utilizing natural language prompts, GLIDE can generate or restore images with contextually accurate details, pushing the boundaries of image editing and restoration.

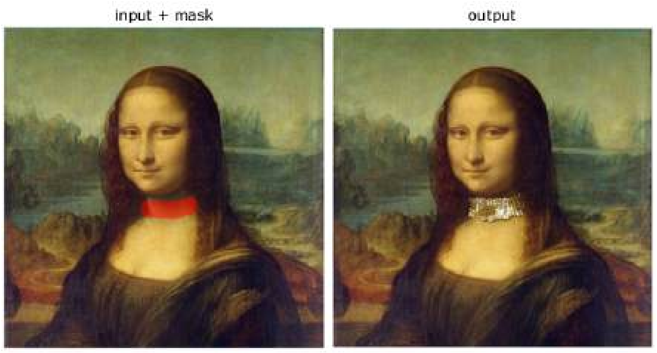


Fig. 3. An example of inpainting with GLIDE. The model receives an input as the Mona Lisa as well as a mask and a text description "a golden necklace". Then, it generates the output as Mona Lisa with a golden necklace. [6].
.

B. Performance Evaluation and Quantitative Analysis

To evaluate GLIDE's effectiveness in image inpainting, we use two quantitative metrics: PSNR and SSIM.

- **PSNR**: Measures pixel-level difference between the original and inpainted images, with higher values indicating better quality. The equation for PSNR is:

$$\text{PSNR} = 10\log_{10}\left(\frac{R^2}{MSE}\right)$$

where R is the maximum possible pixel value of the image (e.g., 255 for an 8-bit image), and MSE is the Mean Squared Error between the original and inpainted images.

- **SSIM**: Assesses perceptual similarity, focusing on structural integrity, with values from -1 to 1 (1 being perfect similarity). The equation for SSIM is:

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

where:

- μ_x and μ_y are the mean pixel values of the original and inpainted images, respectively.
- σ_x^2 and σ_y^2 are the variances of the original and inpainted images, respectively.
- σ_{xy} is the covariance between the original and inpainted images.
- C_1 and C_2 are small constants to stabilize the division (typically $C_1 = 0.01^2$ and $C_2 = 0.03^2$).

By using both PSNR and SSIM, we obtain a more comprehensive evaluation of GLIDE's performance in inpainting tasks, as these metrics collectively capture both pixel accuracy and structural fidelity.

IV. Concept to Code

This section provides a complete guide from the initial dataset setup to running the inpainting model using GLIDE. Each step explains the concept and provides practical instructions to implement the inpainting process in a Colab environment.

Access the code at `https://github.com/MHC-FA24-CS341CV/beyond-the-pixels-emerging-computer-vision-research-topics-fa24/blob/main/code/08-im-inpainting/GLIDE_Inpainting.ipynb`.

1) **Dataset:** This GLIDE inpainting project uses datasets containing images with specific regions masked out, simulating real-world inpainting scenarios, such as restoring parts of faces or objects in natural scenes.
2) **Source:** Images can be uploaded directly into the Colab environment. Common sources for similar tasks include CelebA (for face images) and other standard computer vision datasets for diverse objects and scenes.
3) **Key Characteristics:**
   - The dataset contains images of varied subjects, making it suitable for evaluating GLIDE's inpainting capabilities across different contexts.
   - Specific regions of each image are masked, allowing GLIDE to restore or complete these parts based on surrounding context.
4) **Clone Repository:** Clone the official GLIDE repository to access the model and inpainting functions. Use the following commands to clone and navigate to the repository:

```
!git clone https://github.com/openai/glide-text2im.git
%cd glide-text2im
```

5) **Set Up Environment:** Install the necessary libraries and set environment paths as shown in the Colab notebook. This ensures that all dependencies required for GLIDE are available.
6) **Upload and Process Image:**
   - Upload an image to Colab that you wish to use for inpainting.
7) **Run Inpainting:**
   - Adjust parameters like `batch_size` (default 1), `guidance_scale` (default 3), and `upsample_temp` (default 0.997) to control inpainting quality and style.
   - Run the model to generate inpainted outputs, filling in masked regions based on surrounding content and set parameters.
8) **Calculate Metrics (Optional):** After generating the inpainted image, calculate PSNR and SSIM by comparing it to the original image.

V. Testing and Analysis

A. Goal

The primary goal of this evaluation is to assess the performance of the inpainting model across prompts of varying complexity. The quality of the inpainted results is measured using Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM).

B. Methodology

To test the model's adaptability, three prompts of increasing complexity were used:

- **Easy**: Add two chairs to the scene.
- **Medium**: Add a dog lying down under the chairs to the scene.
- **Hard**: Place a fountain in the scene, introduce ambient lighting from the top-left, and change the wall color to light blue with a floral pattern.

Each inpainted image was evaluated against the original image using PSNR and SSIM metrics to measure pixel similarity and structural accuracy.

C. Results

In Figure 4, the left side shows the original image with a dog sitting between chairs in a café-like setting, providing the context for inpainting. The right side displays the masked image, where a solid gray area covers the dog and parts of the surrounding chairs. This masked region is where inpainting occurs, with the model tasked to fill in this area based on surrounding details, seamlessly blending it with the unmasked parts of the image.

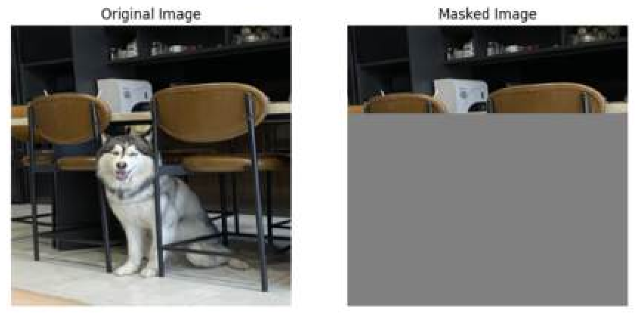


Fig. 4. Original image and masked region for testing. The gray area represents the masked section, which the model attempts to reconstruct.

Table I summarizes the PSNR and SSIM values for each prompt level. Figures 5, 6, and 7 illustrate the original and inpainted images for each prompt.

TABLE I
PSNR AND SSIM VALUES FOR EACH PROMPT COMPLEXITY

Prompt Complexity	PSNR (dB)	SSIM
Easy	29.73	0.39
Medium	29.61	0.36
Hard	29.60	0.36

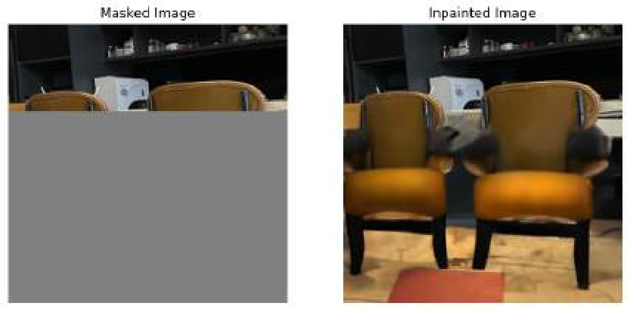


Fig. 5. Inpainting result for the Easy prompt: "Add two chairs to the scene."

For the **Easy** prompt (Figure 5), the model achieved a PSNR of 29.73 dB and an SSIM of 0.39, indicating a high level of pixel similarity. The inpainting result shows a smooth background replacement, consistent with surrounding textures, as object removal is generally simpler for the model.

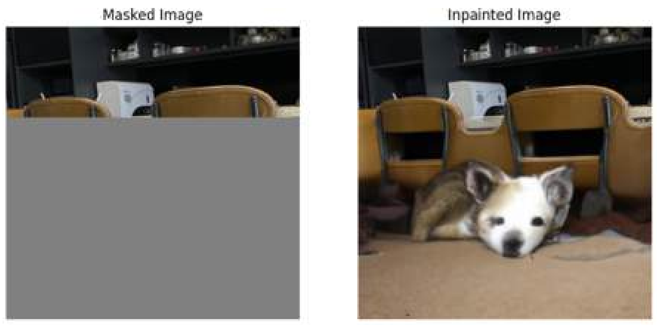


Fig. 6. Inpainting result for the Medium prompt: "Add a dog lying down to the scene."

For the **Medium** prompt (Figure 6), the PSNR was 29.61 dB with an SSIM of 0.36, reflecting a moderate structural change. The model partially succeeded by repositioning the dog's body, but the face and overall shape lack realistic detail, with some distortions present.

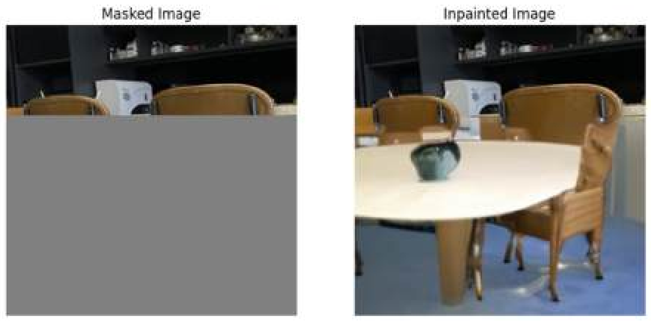


Fig. 7. Inpainting result for the Hard prompt: "Place a fountain in the scene, introduce ambient lighting from the top-left, and change the wall color to light blue with a floral pattern."

For the **Hard** prompt (Figure 7), the PSNR remained at 29.66 dB, similar to the Easy prompt, indicating minimal pixel-level change. However, the SSIM of 0.37 shows the model's difficulty in preserving structural similarity. The model produced a distorted shape instead of a realistic fountain and lacked an accurate background change, failing to apply the expected wall color and floral pattern, which reveals its limitations in handling complex, multi-layered tasks.

VI. Open Challenges and Current Limitations

In the field of image inpainting, there are several open challenges and limitations that researchers continue to address:

- **Resolution limitations:** GLIDE's output resolution is limited, with images being downsized to 64x64 during processing and only able to be upsampled to 256x256 without introducing artifacts. Higher resolution outputs are challenging to achieve without upsampling techniques [4].
- **Handling complex prompts:** While GLIDE can render a wide variety of text prompts zero-shot, it can have difficulty producing realistic images for more complex prompts [4].
- **Contextual consistency:** Maintaining contextual coherence between the inpainted region and surrounding image content remains challenging, particularly for complex images with intricate details [7].

VII. Research Idea

This research proposes enhancements to GLIDE's image inpainting capabilities, focusing on improving performance in complex scenes. Potential improvements include:

- **Dynamic Resolution Upsampling:** GLIDE's current upsampling can cause artifacts beyond 256x256 resolution. Integrating a progressive upsampling approach, leveraging super-resolution techniques, could enhance photorealism, making GLIDE more suitable for applications like large-format photo restoration and high-res satellite imaging.

- **Evaluation Framework with Multi-Dimensional Metrics:** To better assess inpainting success, we can go beyond PSNR and SSIM by adding perceptual metrics like FID (Fréchet Inception Distance) and LPIPS (Learned Perceptual Image Patch Similarity). This multi-metric approach offers a fuller performance view, especially for applications where perceptual quality matters as much as pixel accuracy, such as in artistic restoration and autonomous driving.

These enhancements aim to address GLIDE's current limitations in handling detailed textures, maintaining contextual coherence, and ensuring consistent lighting across the inpainted regions.

REFERENCES

[1] "Image Inpainting — link.springer.com," https://link.springer.com/referenceworkentry/10.1007/0-387-30038-4_98, [Accessed 06-11-2024].

[2] R. Mitson, "Introduction to image inpainting with deep learning — wandb.ai," https://wandb.ai/wandb_fc/articles/reports/Introduction-to-image-inpainting-with-deep-learning--Vmlldzo1NDI3MjA5, [Accessed 06-11-2024].

[3] S. Zarif, I. Faye, and D. Awang Rambli, "Image completion: Survey and comparative study," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 29, p. 1554001, 12 2014.

[4] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," 2022. [Online]. Available: https://arxiv.org/abs/2112.10741

[5] "Generating and editing photorealistic images from text-prompts using OpenAI's GLIDE — blog.paperspace.com," https://blog.paperspace.com/glide-image-generation/, [Accessed 10-11-2024].

[6] A.-S. Maerten and D. Soydaner, "From paintbrush to pixel: A review of deep neural networks in ai-generated art," 02 2023.

[7] "ecva.net," https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136770564.pdf, [Accessed 08-11-2024].

Enhancing Human Pose Estimation: The Role of DeepPose as a Holistic Method and the Potential of End-to-End Learning Approaches

***Abstract*— This project explores two prominent human pose estimation models: DeepPose and Learning Human Pose Estimation Features with Convolutional Networks. The goal is to evaluate the effectiveness of these models in estimating human body keypoints from images and to investigate the feasibility of applying an end-to-end learning approach to the DeepPose model. We examine how the end-to-end approach can streamline pose estimation by eliminating the need for intermediate steps, such as manual feature extraction or multi-stage processing. Through comparative analysis, we demonstrate that end-to-end learning can improve model performance and efficiency, offering a promising direction for future research in human pose estimation.**

I. Introduction

Human Pose Estimation (HPE) is a key task in the field of computer vision aimed at detecting and localizing human joints from images. The problem can be approached using deep learning methods that leverage convolutional neural networks (CNNs) and Deep Neural Networks (DNNs).

we can group deep learning methods in human pose estimation into holistic and part-based methods [1]. DeepPose is a holistic model, and the other one is a part-based method [1].

II. DeepPose

A. Introduction

The DeepPose model represents a holistic approach to human pose estimation, addressing the challenge of estimating human body joint locations, even when some joints are occluded or not visible in the image [2]. In scenarios where certain body parts are hidden from view, as one can see in Fig.1, holistic reasoning becomes crucial for accurately predicting the positions of these occluded joints [2]. The pose estimation task is framed as a DNN-based regression problem, where the network learns to directly predict the locations of body joints [2]. This formulation offers two key advantages:

1) The DNN model has shown outstanding performance on object localization [3]. It is able to capture the global context surrounding each joint, improving the accuracy of joint localization [2].
2) The DNN-based approach provides a more straightforward and less computationally complex alternative compared to methods that rely on graphical models or other structured prediction techniques [2].

These benefits make the DeepPose model an appealing solution for human pose estimation tasks, especially in environments where occlusion is common.

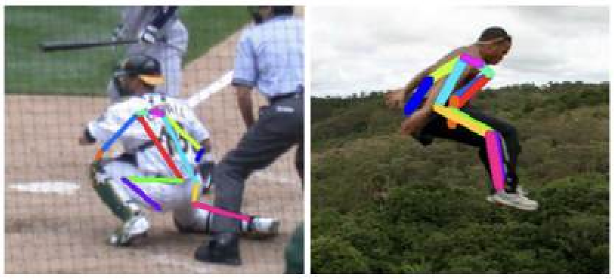


Fig. 1. Many joints are barely visible in certain images, requiring holistic reasoning to predict their locations based on the visible parts of the body and the person's motion or activity [2].

B. Metrix

To facilitate comparison with results, we will use two widely recognized evaluation metrics. The Percentage of Correct Parts (PCP) measures the detection rate of limbs, where a limb is considered detected if the distance between the predicted joint locations and the true joint locations is no more than half the length of the limb [4]. PCP was initially the preferred metric, but it has the drawback of penalizing shorter limbs, such as the lower arms, which are typically more challenging to detect [2].

To address this limitation, a newer metric has emerged, which evaluates joint detection rates using an alternative criterion [2]. In this approach, a joint is considered detected if the distance between the predicted and true joint locations is within a specific fraction of the torso diameter [2]. By adjusting this fraction, detection rates can be reported for different levels of localization precision [2]. This metric, known as Percent of Detected Joints (PDJ), mitigates the issues associated with PCP by applying a consistent detection threshold across all joints [2].

C. Functionality

Basically DeepPose is an approach for human pose estimation using deep neural networks (DNNs), specifically focusing on how pose estimation is framed as a regression problem and how the model is trained and refined using a cascade of pose regressors, and each cascade stage progressively improves the joint locations [2].

Stage 1 starts by using the full image or a person detector to crop out a bounding box around the human subject [2]. The first DNN regressor predicts the joint locations [2].

Subsequent Stages (Stage 2 and beyond) focus on refining the joints by cropping out regions around the predicted joint locations from the previous stage [2]. This allows the

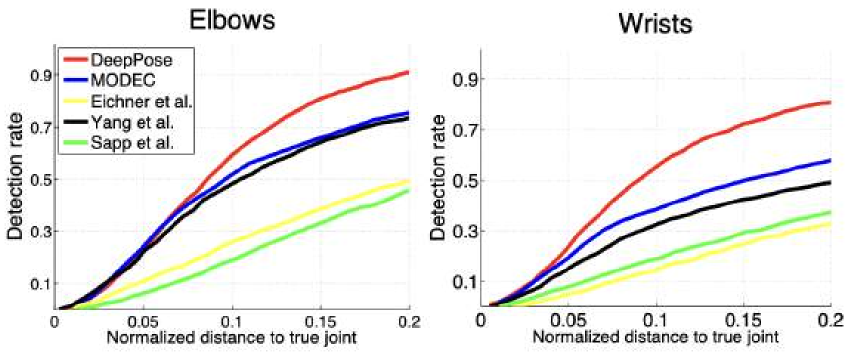


Fig. 2. Percentage of Detected Joints (PDJ) on the FLIC dataset for the elbow and wrist joints. We compare the performance of DeepPose, after two cascade stages, with four other methods [2].

network to focus on high-resolution details of the body parts, improving localization precision [2]. At each stage, a new network is trained to predict the displacement of the joint from its previous stage estimate. These refined locations are then used to generate sub-images for the next stage, improving localization at finer scales [2].

Please refer to Table I for an example. It presents the results from the previous three stages, where a clear trend is observed: an increasing number of stages leads to a higher PCP percentage.

D. Evaluation

The results we obtained on the FLIC dataset are shown in Fig.2, where we compare our method with four other approaches. Our improvements are particularly noticeable in the low-precision domain, where the focus is on detecting approximate poses rather than precisely localizing the joints [2]. At a normalized distance of 0.2 on the FLIC dataset, we observe a 0.15 increase in detection rate for the elbow and a 0.2 increase for the wrists compared to the next best-performing method [2]. This demonstrates that, as a holistic approach, DeepPose outperforms others in general human pose estimation.

Method	Arm		Leg		Ave.
	Upper	Lower	Upper	Lower	
DeepPose-st1	0.5	0.27	0.74	0.65	0.54
DeepPose-st2	0.56	0.36	0.78	0.70	0.60
DeepPose-st2	0.56	0.38	0.77	0.71	0.61

TABLE I
PERCENTAGE OF CORRECT PARTS (PCP) AT 0.5 ON LSP(ANOTHER DATASET) FOR DEEPPOSE [2].

III. END TO END INSPIRATION

Now, let's explore the part-based method. This approach involves training several smaller CNNs for independent binary classification of body parts, followed by a higher-level weak spatial model that helps eliminate significant outliers and ensures global pose consistency [1].

One of the key limitations of earlier methods for tackling full pose estimation using end-to-end learning approaches, particularly deep networks, was the scarcity of labeled data [5]. It is worth noting that the authors, who proposed this part-based method, address the problem of limited labeled data for pose estimation by using larger, more detailed dataset and by making better use of the rich structure of pose annotations to train better models [5]. The dataset used here is FLIC.

Given that DeepPose is also capable of working with the FLIC dataset, this presents an interesting opportunity to explore how end-to-end learning could be implemented within DeepPose as well. The rich and structured annotations in FLIC could potentially enable a more holistic approach to pose estimation, bypassing the need for manual feature engineering. This leads us to believe that, with the right adjustments, DeepPose could benefit from a similar strategy of leveraging larger datasets and sophisticated annotations to better train a model using end-to-end learning. By integrating these techniques into DeepPose, we might not only improve pose detection accuracy but also enhance the model's ability to generalize across different poses, potentially closing the gap between part-based methods and fully end-to-end systems.

IV. CONCLUSION

DeepPose, as a holistic method, holds a unique and irreplaceable role in human pose estimation, particularly in its ability to effectively estimate joints that are not visible in the image. The results demonstrate a distinct advantage in detecting approximate poses, which, in certain practical applications, may prove to be more important than precisely localizing individual joints. This advantage lies in DeepPose's ability to infer the global human pose from partial or occluded body parts, making it robust to variations in pose visibility.

However, one limitation of DeepPose is its lack of research into the application of end-to-end learning methods. In contrast, another study on a part-based method that incorporates end-to-end learning shows promising results, suggesting that this approach could enhance pose estimation performance by directly learning the mappings from input images to joint locations without relying on manual steps.

References

[1] A. D. E. P. Athanasios Voulodimos, Nikolaos Doulamis. (2018) Deep learning for computer vision: A brief review. Accessed: 2024-11-16. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1155/2018/7068349

[2] A. Toshev and C. Szegedy. (2014) Deeppose: Human pose estimation via deep neural networks. USA. 2-s2.0-84911381180. [Online]. Available: https://doi.org/10.1109/CVPR.2014.214

[3] C. Szegedy, A. Toshev, and D. Erhan. (2013) Object detection via deep neural networks. [Online]. Available: https://papers.nips.cc/paper/2013

[4] M. Eichner, M. Marin-Jimenez, A. Zisserman, and V. Ferrari, "Articulated human pose estimation and search in (almost) unconstrained still images," ETH Zurich, D-ITET, BIWI, Tech. Rep. Technical Report No. 272, 2010. [Online]. Available: https://www.biwi.ethz.ch/

[5] A. Jain, J. Tompson, and M. Andriluka. (2014) Learning human pose estimation features with convolutional networks. [Online]. Available: https://openreview.net/forum?id=ryxriwHeZ

The Evolution and Challenges of OCR Technology

Sadichchha Maharjan

Abstract— **OCR (Optical Character Recognition) refers to the process of extracting data from scanned paper documents and images and converting it to machine-readable text. This paper explores the development, current state, and applications of OCR. This paper presents a review of the state-of-the-art methods leveraging deep learning while also highlighting the limitations these approaches encounter. Finally, the paper proposes potential research directions to improve OCR's accuracy in recognizing characters from multilingual texts.**

Index Terms— **Optical Character Recognition; Bilingual Texts; Computer Vision**

I. Introduction

The development of OCR has substantially reduced the amount of time required to digitalize texts and documents and has seamlessly become an integral part of various fields. OCR technology has transformed how we interact with and access information, from reading the license plate numbers of cars on the highways to translating street signs in real-time using a phone camera to providing blind or visually impaired people with the ability to scan printed text and then have it read back to them.

A. Brief History

OCR technology began in the early 20th century with Dr. Gustav Tauschek's rudimentary system for reading printed text. Significant progress was made in the 1950s when Ralph L. Koster at Bell Labs developed an electronic OCR system using photocells to recognize printed characters. The 1970s and 1980s saw the introduction of template-matching algorithms, which improved OCR's accuracy in industries like banking and postal services.

In the 1990s, Ray Kurzweil revolutionized OCR by combining it with text-to-speech technology, aiding visually impaired individuals. The 21st century ushered in the use of machine learning, particularly deep learning methods like Convolutional Neural Networks (CNNs), which have improved OCR's ability to handle complex fonts and handwriting. Today, OCR is widely used in applications such as digitizing books and recognizing text in street signs.

II. Existing Methods

The first step to OCR software is acquiring the image to extract text from and converting it into a black-and-white version where the dark portions are categorized as characters and the light portions as the background. Then, this image is processed to to remove any spots, fix the alignment and identify the script in it. Characters are then recognized using either of the following methods: [1]

1) Pattern Recognition: Pattern Recognition works by isolating a character and recognizing it by comparing it with the stored glyphs (examples of text in various fonts and formats). This method works best for images that have been written or typed in a font that the program has been previously trained on. [2]

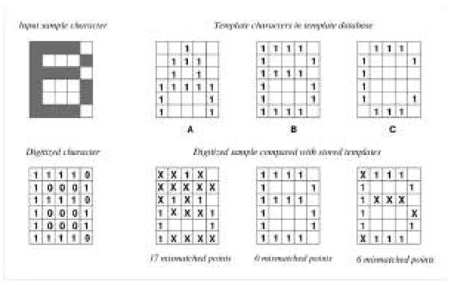


Fig. 1. Illustration of the matching of digitized characters through pattern recognition [3]

2) Feature Recognition: This method is usually used for fonts that the program has not been trained on. It breaks down the input glyphs into features such as lines, intersections, loops, and curves and then uses them to identify the best match within the previously stored glyphs.

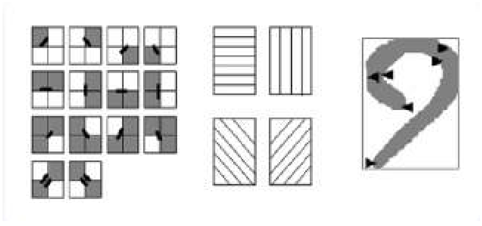


Fig. 2. Illustration of feature extraction from a character [4]

Once the text has been analyzed, the system converts it into a digital file. Some OCR systems can also generate annotated PDF files, preserving both the original and converted versions of the document for easy reference and comparison.

If the described steps work successfully, an OCR software outputs accurate results. However, characters can sometimes be too hard to recognize. Thus, OCR requires more than computer vision technologies. In the recent years, the following technologies have been integrated with OCR to better its performance:

1) Natural language processing (NLP): Even though OCR identifies characters, those characters form words, sentences and paragraphs. Research in NLP has resulted in numerous algorithms that can be used to correct mistakes in character recognition using probabilistic approaches. For example, despite missing characters can be estimated using context.

2) Supervised deep learning: OCR leverages deep learning algorithms to improve its performance. While it requires learning from training samples to improve OCR performance, with this technology, OCR tools can recognize characters with different fonts: Each character can be written in a wide range of forms, and large labelled data set help OCR software identify the characters despite font variations detect errors and correct them. OCR tools can skip characters that cannot be identified. By recognizing patterns in training samples, OCR can detect those errors and correct its mistakes.

III. Open Challenges

Current Limitations of OCR:

1) Handwriting Recognition
2) Complex Document Layouts
3) Multilingual and Bilingual Text:

IV. Concept to Code

This section explores the current abilities of the Pytesseract library to recognize characters from images with multilingual texts. Tesseract has difficulty accurately recognizing text in multilingual documents, especially when multiple languages or scripts are present within the same document, as it struggles to switch between language models effectively.

Dataset to use:

Demonstrate the low accuracy for low-resource languages and complex script combinations

References

[1] Aug 2024. [Online]. Available: https://www.ibm.com/think/topics/optical-character-recognition

[2] [Online]. Available: https://aws.amazon.com/what-is/ocr/

[3] N. Li, "An implementation of ocr system based on skeleton matching," 1993. [Online]. Available: https://api.semanticscholar.org/CorpusID:9931160

[4] G. Tawde, M. J. M. Kundargi, and J. M. Kundargi, "An overview of feature extraction techniques in ocr for indian scripts focused on offline handwriting," 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:11629801

Integrating Motion-Aware and Direction-Aware Techniques for Improved 3D Gaussian Splatting in Dynamic Scenes

Mai Bui

***Abstract*— Accurately reconstructing 3D environments, especially with moving objects, remains a significant challenge. While advanced techniques like neural rendering (NeRF) produce detailed 3D models, they are computationally intensive, limiting real-time applications. 3D Gaussian Splatting (3D GS) offers a more efficient alternative, but struggles with motion blur in dynamic scenes. This work presents an integrated approach that combines motion-aware and direction-aware methods to enhance 3D GS. The motion-aware component uses optical flow to track object movements, while the direction-aware DaRePlane technique captures complex geometries and reduce motion blur. These are integrated through a hierarchical pipeline - first performing a coarse motion-aware reconstruction, then refining it using direction-aware processing. Adaptive splat representations and parallel processing optimize computational and storage requirements, ensuring real-time performance. Experiments show our integrated solution outperforms standalone techniques, reducing motion blur and preserving geometric fidelity, advancing the state-of-the-art in practical 3D scene reconstruction.**

***Index Terms*— Gaussian Splatting ; motion-aware; direction-aware, optical flow, DaRePlane**

I. Introduction

While dynamic scene reconstruction plays an important role in medical training, autonomous driving, sport and performance exhibition, etc, there has been multiple challenges including real-time performance, scene complexity, data requirement and temporal consistency. In this paper, I want to focus on solving the problem of temporal consistency, which means the consistency of data generated at different times.

II. Existing methods & current limitations

A. Neural Radiance Fields (NeRF)

Introduced in 2020, this neural network-based approach was a breakthrough and a great benchmark for synthesizing novel views of complex scenes. It trained multiple images from different viewpoints, going through ray marching and traditional volumn rendering technique to predice volumn density and RGB color for each pixel to create 3D reconstruction of the scene [1]. Since then, there has been significant efforts to improve, producing multiple variants for different purposes: D-NeRF for dynamic scene reconstruction [2], Instant RGB for realtime rendering [3], Nerflies for handling deformable scene [4]. However, despite notable development, its time and power consuming presents great challenges. [1]. Additionally, as NeRF is frame-by-frame based training and is lack of ”recall” previous frame mechanism, this state-of-the-art approach produces noticeable artifacts in dynamic scenes, resulting in bad temporal inconsistency.

Mai Bui with Mount Holyoke College, Department of Computer Sciece, 50 College Street, South Hadley, MA USA {bui23m}@mtholyoke.edu

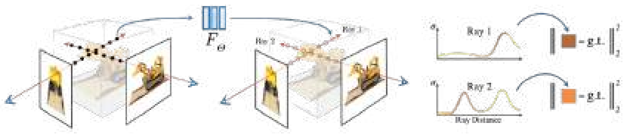


Fig. 1. Neural Radiance Fields (NeRF) Architecture and View Synthesis Workflow [1]

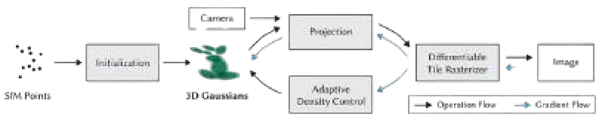


Fig. 2. Neural Radiance Fields (NeRF) Architecture and View Synthesis Workflow [5]

B. Gaussian Splatting (GS)

In order to offer realtime rendering and training which provides tremendously more practical use, 3D GS was introduced as an alternative way for 3D reconstruction. In stead of representing 3D scenes with neural fields, it uses a collection of anisotropic 3D Gaussian functions (splats). Those include information: position, covariance, color, opacity. By projecting 3D images into 2D planes, then went through point-based volumn rendering, this approach produced photorealistic images from any viewpoint [5]. However, due to its reliance on splats, in fast-moving environment, splats may shift or distort, resulting in blurry or jittery reconstruction.

C. Motion-aware

In order to solve the loss of motion information in fast paced environment, optical flow, in motion-aware methods was introduced to track object movement across frames, aiding in reducing temporal inconsistencies. However, conventional optical flow methods struggle with covariance shifts and lack direction sensitivity, limiting their effectiveness in capturing complex motions. [6]

D. Direction-aware

To overcome problem with direction sensitivity, DaRePlane introduced a direction-aware representation that captures 6 instead of 3 traditionally directions. This helps the drop from 4D to 2D less steep, lossing less information, capturing detailed information in dynammic settings and producing superior performance in novel view synthesis. [7]

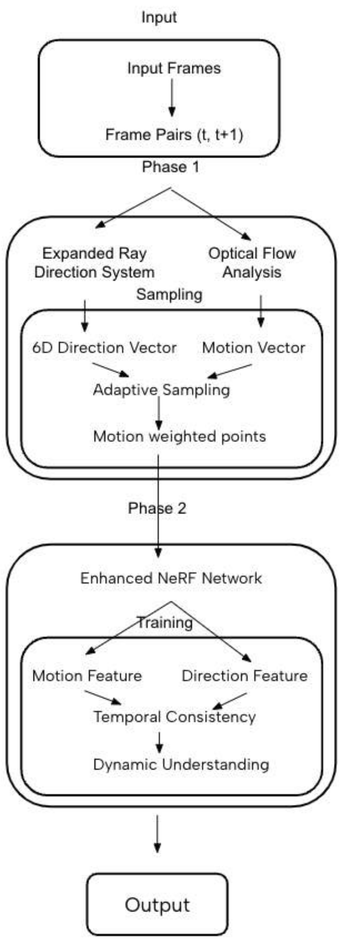


Fig. 3. Implementation plan coding workflow

III. Research Proposal

A. Integration Idea

I am interested in integration direction-aware and motion-aware methods and put it into Gaussian Splatting to evaluate the model's improvement easier and more holistic. I believe the benefit of getting detailed information from direction-aware will be a great addition/component to overcome the motion blur in motion-aware. And given motion information will contribute to provide better reasoning for transparancy and occultation of motion-aware approach. In the future, I would love to present a pipeline to ensure smooth integration between 2 approaches and motion, direction-aware with 3D GS and NeRF. In this proposal, as Google Colab is an environment with limit of GPU usage and not suitable for 3D GS's complex dependency, I focus on creating pipeline and planning for integration in NeRF.

B. Code explanation

Code can be found in this Github Link

1) Phase 1: Direction and Motion Enhancement: The first phase will focus on expanding the ray direction system from three to six dimensions, integrating up/down viewing angles and forward/backward/left/right motion predictions alongside existing spatial coordinates. This foundational change will require careful modification of the get_rays function while maintaining compatibility with existing rendering pipelines. Once the expanded directional system is stable, I'll implement motion tracking capabilities through optical flow analysis between consecutive frames - this represents a critical bridge between static and dynamic scene understanding.

More specifically, I'll create a preprocessing stage that analyzes pairs of consecutive frames to compute motion vectors - these vectors will track how each point moves from one frame to the next. A new preprocessing stage analyzes consecutive frame pairs to compute motion vectors, tracking point movement between frames. This requires the implementation of a compute_optical_flow function, which can utilize either traditional computer vision techniques or a specialized neural network for pixel-level movement estimation. Real-time performance remains a critical consideration throughout this implementation.

Building upon this foundation, I introduce an adaptive sampling strategy that revolutionizes how the system processes dynamic scenes. The render_flat_rays function undergoes enhancement to adjust sampling density based on detected motion. For instance, when capturing a scene with a moving person, the system intelligently allocates more computational resources to areas with significant movement while maintaining efficient processing of static backgrounds. This adaptive approach relies on a sophisticated weighting algorithm that considers both motion magnitude and direction.

2) Neural Network Enhancement: The second phase focuses on architectural modifications to the neural network itself. The network requires significant redesign to accommodate six directional dimensions and specialized layers for motion flow features. This enhancement enables new predictive capabilities, including velocity and acceleration estimation, along with uncertainty quantification for moving objects within scenes.

The training pipeline receives corresponding updates to support these new capabilities. Temporal consistency checks and motion-aware loss functions ensure the system learns to accurately predict and represent dynamic scene elements. To manage the increased computational complexity, I implement progressive training strategies that carefully balance performance requirements with available resources.

C. Experiments

With the theoretical foundation established, our next steps focus on practical implementation and evaluation. The immediate priority is to implement the enhanced NeRF system, starting with the expanded direction-aware components followed by motion tracking integration. I plan to conduct comprehensive experiments comparing our enhanced model against baseline NeRF implementations, particularly focusing on scenes with significant dynamic elements and complex viewing angles and using Plenoptic Video Dataset [8] and Cochlear Implant Surgery [9].

TABLE I
OVERVIEW OF EXPERIMENTAL DESIGN AND METRICS

Aspect	**Details**	**Evaluation Metric**
Dataset	Plenoptic Video Dataset, Cochlear Implant Surgery Dataset	Robustness in dynamic and medical scenes
Baseline	Vanilla NeRF implementations	Motion handling, view rendering
Enhancements	Direction-aware system, motion tracking integration	PSNR, SSIM, temporal consistency
Qualitative	Visual motion and view effects	Dynamic scene realism
Future Work	Transition to Gaussian Splatting	Rendering quality, efficiency

D. Evaluation

My evaluation metrics will include both quantitative measures (PSNR, SSIM, temporal consistency scores) and qualitative assessments of motion handling and view-dependent effects. I anticipate that initial results from these experiments will provide valuable insights for further optimization and refinement of the integration approach.

IV. FUTURE DIRECTION

Looking ahead, I am particularly excited about transitioning this enhanced approach to Gaussian Splatting. Given Gaussian Splatting's superior performance in terms of rendering quality and training efficiency, I believe our direction and motion-aware enhancements could yield even more impressive results in this context. While this transition will present its own technical challenges - particularly in adapting our motion tracking and directional encoding to work with Gaussian primitives - the potential benefits in terms of rendering quality and computational efficiency make this a compelling next step. The lessons learned from our NeRF implementation will provide crucial insights for this future adaptation, potentially leading to even more efficient and effective dynamic scene representation methods.

REFERENCES

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf," *Communications of the ACM*, vol. 65, no. 1, p. 99–106, Dec 2021.

[2] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-nerf: Neural radiance fields for dynamic scenes," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 10313–10322, Jun 2021.

[3] S. Rojas, J. Zarzar, J. C. Pérez, A. Sanakoyeu, A. Thabet, A. Pumarola, and B. Ghanem, "Re-rend: Real-time rendering of nerfs across devices," *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, p. 3609–3618, Oct 2023.

[4] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla, "Nerfies: Deformable neural radiance fields," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, p. 5845–5854, Oct 2021.

[5] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, p. 1–14, Jul 2023.

[6] Z. Guo, W. Zhou, L. Li, M. Wang, and H. Li, "Motion-aware 3d gaussian splatting for efficient dynamic scene reconstruction," 2024. [Online]. Available: https://arxiv.org/abs/2403.11447

[7] A. Lou, B. Planche, Z. Gao, Y. Li, T. Luan, H. Ding, M. Zheng, T. Chen, Z. Wu, and J. Noble, "Dareplane: Direction-aware representations for dynamic scene reconstruction," 2024. [Online]. Available: https://arxiv.org/abs/2410.14169

[8] T. Li, M. Slavcheva, M. Zollhoefer, S. Green, C. Lassner, C. Kim, T. Schmidt, S. Lovegrove, M. Goesele, R. Newcombe, and Z. Lv, "Neural 3d video synthesis from multi-view video," 2022. [Online]. Available: https://arxiv.org/abs/2103.02597

[9] A. Lou, X. Yao, Z. Liu, J. Han, and J. H. Noble, "Self-supervised surgical instrument 3d reconstruction from a single camera image," *Medical Imaging 2023: Image-Guided Procedures, Robotic Interventions, and Modeling*, p. 14, Apr 2023.

Image Colorization with deep convolutional Neural Network

Anika Nazhat
Mount Holyoke College
nazha24a@mtholyoke.edu

***Abstract*— This project explores Image Colorization using a Convolutional Neural Network (CNN)-based encoder-decoder architecture. The model takes colored galaxy images as input, where the primary task is to predict the missing color channels (a and b) in the Lab color space. The network is trained on a dataset consisting of colored galaxy images, learning to extract spatial features and associate them with accurate color predictions. The model is trained for several epochs, where the grayscale L channel of the Lab color space is used as input and the model learns to predict the chrominance channels (a and b). The trained model is evaluated on a set of test images to assess its performance and ability to generalize to new, unseen galaxy images. The aim of this project is to develop a deep learning model that can provide realistic colorization of galaxy images. Space images contain unique color patterns that correspond to various elements and cosmic phenomena. Colorizing grayscale versions of these could help train a model that can detect such phenomena through colorization.**

***Index Terms*— keyword 1; keyword 2; keyword 3**

I. Introduction

Image colorization is the process of transforming grayscale images into color based on the image's content. Colorization has been revolutionized by deep learning, particularly with Convolutional Neural Networks (CNNs). CNNs are well-suited for this task as they extract image features—such as edges and textures—through convolutional layers, allowing the model to analyze grayscale images and generate plausible color predictions by learning from large datasets.

Our approach uses a CNN-based encoder-decoder architecture to predict colors for grayscale images. Our model takes in colorized images. These images are then converted to separate the grayscale (L channel) from the color information (a and b channels). The L channel serves as the model's input. This setup allows the network to learn associations between grayscale patterns and specific color values by training on a dataset of color images, enabling it to generate plausible colorizations for new images. This approach assumes that effective colorization requires understanding both the image's structure and its content, which guides the prediction of realistic color values.

The model is trained on a set of colored images. Initially, the images are converted to the LAB color space where the L(Lightness) colorspace, which is the grayscale version of the images, is taken as the input for the Convolutional Neural networks (CNNs). The model architecture consists of an encoder-decoder structure with convolutional layers, batch normalization, and upsampling techniques. This architecture is designed to capture the spatial features in the input image and use them to reconstruct the missing color information. After training, the model is capable of generating colorized versions of grayscale images where it predicts the A and B colorspace of the images, even those it has never seen before.

The goal of this project is not only to generate aesthetically pleasing colorized images but also to demonstrate how deep learning can be used to tackle a classical problem in image processing. By the end of this project, we aim to understand the strengths and limitations of using a CNN for image colorization, and provide insights into the potential for further improving the quality of automated image colorization methods.

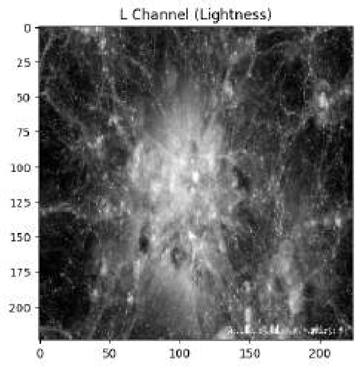


Fig. 1. grayscale (L channel)

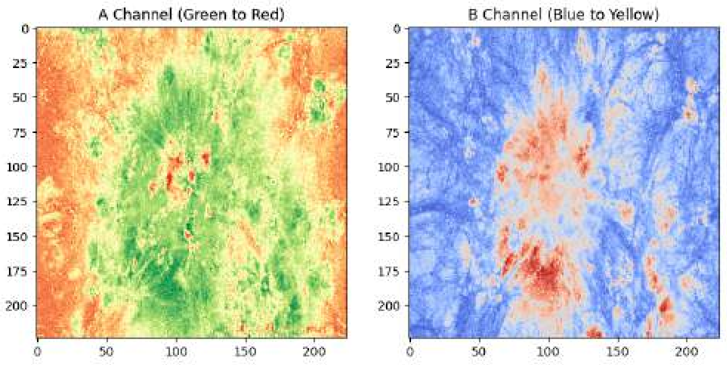


Fig. 2. color information (a and b channels)

II. Existing Methods

The task of image colorization has seen significant advancements, transitioning from interactive systems to fully automated deep learning methods. Early work focused on user-guided techniques, such as grouping pixels into coherent regions for color mapping. For example, Luan et al [1] proposed a method where users group regions with similar textures or intensities, followed by a mapping process to apply vivid colors to each region. This approach reduced user effort compared to purely manual methods but still required substantial interaction, limiting scalability to larger datasets or complex images [1].

Deep learning has greatly transformed the field, with convolutional neural networks (CNNs) emerging as the dominant approach. Hwang and Zhou presented a CNN-based system that utilized the CIELUV color space to separate the luminance (grayscale) and chrominance (color) components of an image. Their model employed a regression-based approach to predict chrominance channels, though they noted limitations in the vibrancy of the generated colors due to averaging effects inherent in their loss function [2].

Addressing the ambiguity in colorization, Deshpande et al [3] developed a method based on variational autoencoders (VAEs) to generate multiple plausible colorizations for a single grayscale input. By learning a low-dimensional embedding of color fields and employing a conditional mixture density network, their approach produced diverse and spatially coherent results, outperforming simpler CNN models in terms of visual realism [3].

The growing availability of large datasets has enabled more automated methods. Techniques discussed in the ECCV proceedings emphasize the importance of high-quality training data and robust architectures, such as encoder-decoder models, to improve generalization to new images. These models leverage features extracted from pretrained networks, often combined with transfer learning to accelerate convergence and improve performance [4] [5].

While traditional methods like those discussed by Luan et al [1]. focused on interactive segmentation and mapping, modern approaches prioritize minimizing user involvement and enhancing the realism of generated colorizations. However, challenges remain in handling high-resolution images, generalizing across diverse datasets, and managing the inherent ambiguity of colorization tasks.

III. Open challenges

Current limitations: Image colorization faces several challenges, with ambiguity being a key issue. Multiple plausible colorizations can exist for the same grayscale image, and while methods like VAEs and CMDNs aim to generate diverse outputs, they often struggle to balance realism with variation. This can result in outputs that are either overly uniform or lack spatial coherence [3] [2].

Another limitation is the difficulty of generalizing across diverse datasets. Models trained on specific datasets often fail to perform well on images with unfamiliar textures or objects. While transfer learning improves feature extraction, it falls short in adapting to unique domains like scientific or high-resolution images. Additionally, computational demands for training and inference at higher resolutions limit scalability, making efficient, accurate high-resolution colorization an ongoing challenge [2] [4].

Research Idea: The research question I am exploring is ”How can deep learning, specifically convolutional neural network (CNN)-based encoder-decoder architectures, be utilized to generate realistic and scientifically meaningful colorizations of grayscale galaxy images?”.

Recent studies, such as Kalvankar et al. (2021) [6], have used Generative Adversarial Networks (GANs) and pretrained ResNet-18 models to automate the colorization of astronomical images, focusing on improving resolution and visual appeal. Similarly, Rector et al. (2007) [7] discussed techniques for combining multiwavelength data to create accurate and visually compelling images [6] [7].

My research differs by aiming to not only colorize galaxy images but also teach the model to identify cosmic phenomena, like star formations, through these colorizations. To achieve this, I will focus on refining the model to learn patterns that connect specific colors with physical features in the images, making the outputs both visually meaningful and scientifically informative.

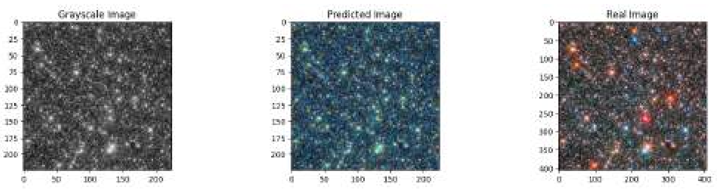


Fig. 3. Grayscale, Predicted and Real Image

IV. Concept to Code

Here is the GitHub source code folder link:
`https://github.com/MHC-FA24-CS341CV/beyond-the-pixels-emerging-computer-vision-research-topics-fa24/tree/main/code/12-im-colorization/FINAL_Image_Colorization_Using_CNN_Anika`

Description of the Dataset(s): The dataset used for this project consists of colored galaxy images, specifically curated for training and testing the colorization model.

Source: Galaxy images are typically sourced from publicly available astronomy datasets, such as those provided by NASA or the Sloan Digital Sky Survey (SDSS). For this project, the dataset was sourced from Kaggle.

Key Characteristics: Images are in RGB format and are converted into the Lab color space during preprocessing. The Lab color space separates lightness (L) from color information (a and b), enabling a focus on grayscale structure for input and chrominance for output. The dataset includes diverse galaxies with unique structures, textures, and color patterns, which provide rich training data for the model to learn realistic and scientifically meaningful colorization.

Run Instructions: To execute the Colab notebook for the project, follow these steps:

Dataset Preparation: Upload the dataset to a Google Drive folder. Ensure images are in a compatible format (jpg) and structured into training and validation directories.

Google Colab Setup: Mount Google Drive by running the provided from google.colab import drive command. Provide the appropriate path to access the dataset from Google Drive. Environment Setup:

Install the necessary libraries such as TensorFlow, NumPy, OpenCV, and any other dependencies listed in the notebook.

Running the Code: Execute the notebook cells sequentially: Data Preprocessing: Converts the dataset to Lab color space and prepares training data. Model Training: Trains the CNN model to predict a and b channels from the L channel. Evaluation: Uses the test dataset to predict colorized outputs of the grayscale images.

Visualizing Results: Generated images can be visualized directly in the notebook or saved back to Google Drive for offline access.

Connection to the Proposed Research Idea: This program supports the research idea of using deep learning to enhance image analysis for scientific purposes, specifically by enabling the colorization of grayscale galaxy images.

Scientific Relevance: Color in galaxy images corresponds to key cosmic phenomena (e.g., star formation regions, chemical composition). By training the model to infer accurate color patterns, this project attempts to train a model to not only colorize images but also detect these phenomena through this process which can be useful in the field of Astronomy.

Next Steps for Technical Implementation: To enhance the model's performance and applicability, the following steps can be taken in the future:

Dataset Expansion: Include larger and more diverse datasets, such as those with different galaxy types and spectral imaging data, to improve the model's generalizability.

Model Refinement: Experiment with advanced architectures like U-Net or GAN-based networks to enhance spatial feature extraction and produce more realistic outputs [5]. Performance Optimization:

Fine-tune hyperparameters such as learning rate, batch size, and training epochs. Implement techniques like data augmentation to make the model more robust to variations.

Integration of Domain Knowledge: Incorporate astrophysical constraints or rules into the model (e.g., expected colors for certain regions) to guide the predictions.

REFERENCES

[1] Q. Luan, F. Wen, D. Cohen-Or, L. Liang, Y.-Q. Xu, and H.-Y. Shum, "Natural image colorization," in *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. Eurographics Association, 2007, pp. 309–320.

[2] J. Hwang and Y. Zhou, "Image colorization with deep convolutional neural networks," Stanford University," Technical Report, 2016.

[3] A. Deshpande, J. Lu, R. A. Yeh, M. Jin, and D. Forsyth, "Learning diverse image colorization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2877–2885.

[4] G. Charpiat, M. Hofmann, and B. Schölkopf, "Automatic image colorization via multimodal predictions," in *Computer Vision–ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part III*, vol. 10. Springer Berlin Heidelberg, 2008, pp. 126–139.

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

[6] S. Kalvankar, B. Elger, H. Zacharia, and E. Tempel, "Astronomical image colorization and upscaling with generative adversarial networks," *arXiv preprint arXiv:2112.13865*, 2021.

[7] T. A. Rector, Z. G. Levay, L. M. Frattare, J. English, and M. Pu'uohau-Pummill, "Image-processing techniques for the creation of presentation-quality astronomical images," *The Astronomical Journal*, vol. 133, no. 2, pp. 598–611, 2007.

Analyzing Performance of SimCLR on Different Image Types

Kira Kaplan
Smith College
kakaplan@smith.edu

***Abstract*— Self-supervised learning is an up and coming field of machine learning that relies on an algorithm to train a model to detect patterns from an unlabeled dataset. It's become popular in recent years due to the expensive, and time consuming process of curating labeled, training datasets. However, whether the current technology is at a level where it can sufficiently be deployed as a generalized model remains to be seen. The Simple Framework for Contrastive Learning for Visual Representations (SimCLR) is one algorithm that has shown substantial promise in it's ability to detect and classify objects, even outperforming it's supervised counterparts in some downstream tasks. In this paper a SimCLR model pre trained on an ImageNet dataset is used to classify sets of images from three different TensorFlow datasets (tf_flowers, imagenette, and cifar100). The model performs best on the imagenette dataset and has slight classification errors on the tf_flowers dataset. However, the model repeatedly fails to classify the lower quality images from the cifar100 dataset. This is indicative that SimCLR and other self-supervised learning algorithms may not yet be suitable for direct deployment and need either adjustments to the algorithm to account for low quality images, or to be fine-tuned on a separate dataset before put into use.**

***Index Terms*— Self-supervised Learning; Image Classification; SimCLR**

I. Introduction

Computer vision applications have revolutionized the way we process information and work through datasets. Researchers are now first turning to machine learning and deep learning algorithms to work through their datasets, greatly reducing the manual processing time previously required. These algorithms can mostly be split into two broad categories: supervised, and unsupervised learning. Supervised learning is an approach characterized by the utilization of labeled data to train a model, while unsupervised learning doesn't require labeled datasets and relies on the model itself to detect it's own patterns in the data. While supervised learning has generally been shown to produce better performing models, it requires manual annotations to be made for large datasets. The amount of annotated training datasets available for use is limited and they are time-intensive and expensive to create. Therefore, unsupervised learning approaches are an alternative that is quickly being improved upon to address this issue.

A. Self-supervised Learning

One form of unsupervised learning is what is known as self-supervised learning (SSL). [1] describes this approach as a semi-automatic process in which the model obtains labels from the data itself to then predict part of the data from other parts. It is considered a "recovery" approach where hidden portions of the data are predicted utilizing visible portions [2]. There are two main steps present in SSL that consist of pretext and downstream tasks.

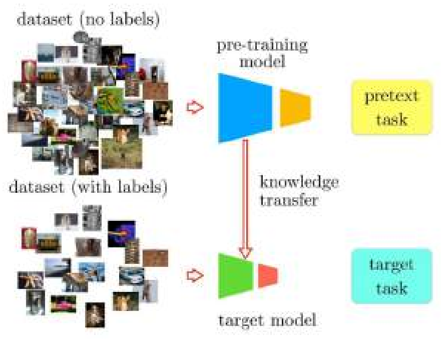


Fig. 1. Fig 1. The basic workflow of self supervised learning. SSL models are trained on unlabeled data and then fine-tuned or tested on their specific downstream test with labeled data [3].

1) Pretext tasks: Pretext tasks are also known as "surrogate" or "proxy" tasks [4] and are a means to achieve the final purpose of the model. They generate pseudo labels for the data through a process P in order to produce a labeled source dataset

$$\bar{D_s} = \{x_i, y_i\} = P(D_s) \qquad (1)$$

where $\bar{D_s}$ represents the original, unlabeled dataset, x_i represents an image in the dataset, and y_i represents the pseudo label generated by the pretext task [5]. There are many different algorithms implemented as pretext tasks, some of which are described later in this paper.

2) Downstream tasks: Downstream tasks are sometimes known as "secondary", "primary", or "target" tasks [2] and define the model's main purpose. These refer to tasks such as object detection, object recognition, image classification, semantic segmentation, natural language processing, and human action recognition to name a few [2], [6], [7].

II. Existing Methods

Existing self-supervised learning algorithms can be split up into four main categories that describe the pretext task used including context-based methods, contrastive learning, generative algorithms, and contrastive generative methods [4].

A. Context-based Methods

Context-based methods include method such as transformation, jigsaw, and colorization that rely on the contextual relationship between samples.

B. Contrastive Learning

Contrastive learning (CL) is focused on measuring the similarity and dissimilarity between images and often relies on methods such as cluster discrimination and instance discrimination [1], [4]. Momentum Contrast (MoCo) and the Simple Framework for Contrastive Learning for Visual Representation (SimCLR) are two CL algorithms that have been shown to outperform their supervised pre-trained model counterparts in some downstream tasks [8], [9].

C. Generative Algorithms

Generative algorithms focus on the reconstruction and are specifically used in masked image modeling (MIM) methods [1], [4]. This involves training the model to fill in missing or masked data based on the patterns it learns itself from the data [5]. Algorithms such as the bidirectional encoder representation from image transformers (BEiT) [6], masked autoencoder (MAE) [10], context autoencoder (CAE) [7], and simple framework for masked image modeling (SimMIM) [11] have all shown great promise in their ability to perform downstream tasks such as object detection, semantic segmentation, and image classification.

D. Contrastive-generative Methods

Contrastive learning and generative algorithms (specifically MIM methods) have been the dominant players in the field [4], but each has their own downfalls. Contrastive models are prone to overfitting issues, while generative models have difficulty with data scaling and have data-filling challenges [12]. This final category for pretext tasks works to combine the best of both these methodologies as a contrastive generative method. It minimizes the contrastive aspects of samples and then reconstructs the inputs [1], drawing from both contrastive and generative methods. One example of this method is in Generative Adversarial Networks (GAN) [13] which involve a two step training process with the generation of fake samples and then an attempt to distinguish them from real samples [1].

TABLE I
EXAMPLES OF SELF-SUPERVISED ALGORITHMS USED FOR THE FOUR MAIN CATEGORIES OF PRETEXT TASKS.

Context-based	Contrastive Learning	Generative	Contrastive-generative
Colorization	SimCLR [9]	BEiT [6]	iBOT [14]
Jigsaw	MoCos [8]	MAE [10]	CMAE [15]
Transformation	Barlow Twins [16]	CAE [7]	SiameseIM [17]

III. Open Challenges

A. Current Challenges

Self-supervised learning alleviates many of the barriers surrounding traditional, supervised machine learning, specifically in regards to creating large, expensive training datasets. However, current limitations still exist in the field of self-supervised learning that may deter future use. One of the major challenges right now in self-supervised learning is determining the best approach to use for a downstream task. An optimal SSL algorithm doesn't exist for a specific task, and the decision usually requires much experimentation and consideration of the dataset to determine the best method [4]. Future research is required to investigate a suitable solution for deciding which SSL method to utilize for the best results without running multiple experiments. Additionally, although some SSL algorithms have been shown to outperform their fellow supervised models [6]–[10], less information exists on how SSL models might perform on less visually distinctive and refined datasets, such as those that experience illusion difficulties and blurriness.

B. Research Ideas

This study provides a preliminary comparison of the performance of SimCLR on different test datasets. Specifically, it looks at the model's ability to run inference on images from a different dataset than it was trained on and on images that may be of reduced quality. The results of this experiment, evaluated qualitatively, will show whether future improvements to SimCLR and other contrastive learning methods are required to achieve performance levels suited for continued use.

IV. Concept to Code

A. Datasets

1) Training Dataset: The SimCLR model used for this study was trained by [9] on the ImageNet ILSVRC-2012 dataset. This is a dataset comprised of images from 1,000 object classes for a total of 1.28 million available images [18]. These images span a wide range of different classes, making the SimCLR pre-trained model ideal as a generalized image classifier.

2) Testing Datatsets: The three example datasets used for testing come from TensorFlow. The tf_flowers dataset is comprised of up-close images of flowers including sunflowers, dandelions, daisies, and tulips [19]. The imagenette dataset is a subset of the ImageNet dataset and contains 10 different classes. The final test dataset is cifar100 which consists of 100 classes and 20 superclasses [20]. These images are significantly more blurred than that of the other two datasets.

B. Running the Code

The code can be run by going to the following GitHub folder: `https://github.com/MHC-FA24-CS341CV/beyond-the-pixels-emerging-computer-vision-research-topics-fa24/blob/main/code/13-self-supervised-learning` and opening the file in Google Colab. Then select “Run All” from Runtime and wait for the process to complete. The outputted images display the models’ predictions alongside the true labels.

C. Results

The SimCLR model performed best at classifying the images from the imagenette dataset (Fig. 2).

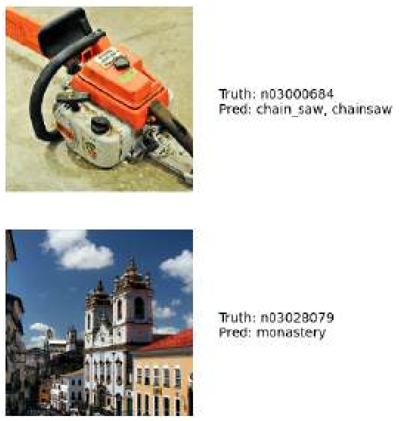


Fig. 2. Results of pretrained SimCLR model classifying images from the imagenette dataset.

This is not surprising since the model was originally trained on the larger ImageNet dataset. This means it might already be familiar with many of the images and the labels of this test dataset. The model had varying results with the tf_flowers dataset. It was generally able to classify an image as some type of flower, although the type of flower was often incorrect (Fig. 3).

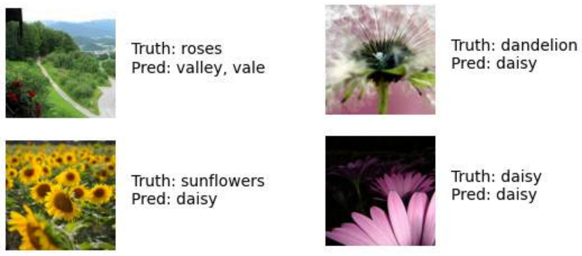


Fig. 3. Results of pretrained SimCLR model classifying images from the tf_flower dataset.

The cifar100 dataset had the worst results, with the model incorrectly classifying almost all of the images. Although most of the classes shown in the test images were similar to those that are labeled in ImageNet, the blurriness of the cifar100 images may have contributed to the model’s inability to identify them (Fig 4.).

This indicates that self supervised learning algorithms such as SimCLR still have a ways to go before they are able to perform as ready to use, generalized models. Avenues of improvement should be investigated to account for low quality images as well as for identifying classes that might not have been seen in the original training dataset.

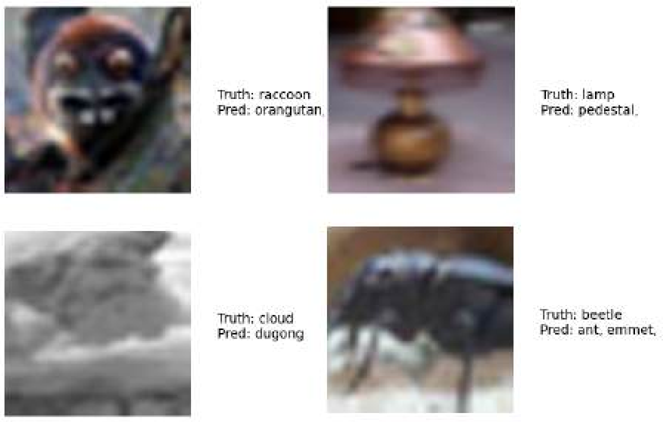


Fig. 4. Results of pretrained SimCLR model classifying images from the cifar100 dataset.

V. Future Work

Next steps may include altering image colorization to see if it impacts the models ability to accurately classify an image. Additionally, more datasets should be included as test subjects to get a more robust sample of SimCLR’s performance. Finally, models with different pretext tasks, such as generative algorithms (MIM methods), contrastive-generative methods, and contex-based methods should be explored with the same test datasets used for SimCLR to compare performances on different image types and styles. This is one step closer to defining an optimal SSL algorithm for different downstream tasks.

References

[1] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, “Self-supervised learning: Generative or contrastive,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, p. 857–876, Jan. 2023.

[2] V. Rani, S. T. Nabi, M. Kumar, A. Mittal, and K. Kumar, “Self-supervised learning: A succinct review,” *Archives of Computational Methods in Engineering*, vol. 30, no. 4, p. 2761–2775, May 2023.

[3] M. Noroozi, A. Vinjimoor, P. Favaro, and H. Pirsiavash, “Boosting self-supervised learning via knowledge transfer,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, June 2018, p. 9359–9367. [Online]. Available: https://ieeexplore.ieee.org/document/8579073/

[4] J. Gui, T. Chen, J. Zhang, Q. Cao, Z. Sun, H. Luo, and D. Tao, “A survey on self-supervised learning: Algorithms, applications, and future trends,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, p. 9052–9071, Dec. 2024.

[5] L. Ericsson, H. Gouk, C. C. Loy, and T. M. Hospedales, “Self-supervised representation learning: Introduction, advances, and challenges,” *IEEE Signal Processing Magazine*, vol. 39, no. 3, p. 42–62, May 2022.

[6] H. Bao, L. Dong, S. Piao, and F. Wei, “Beit: Bert pre-training of image transformers,” no. arXiv:2106.08254, Sept. 2022, arXiv:2106.08254. [Online]. Available: http://arxiv.org/abs/2106.08254

[7] X. Chen, M. Ding, X. Wang, Y. Xin, S. Mo, Y. Wang, S. Han, P. Luo, G. Zeng, and J. Wang, “Context autoencoder for self-supervised representation learning,” no. arXiv:2202.03026, Aug. 2023, arXiv:2202.03026. [Online]. Available: http://arxiv.org/abs/2202.03026

[8] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” no. arXiv:1911.05722, Mar. 2020, arXiv:1911.05722. [Online]. Available: http://arxiv.org/abs/1911.05722

[9] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," no. arXiv:2002.05709, July 2020, arXiv:2002.05709. [Online]. Available: http://arxiv.org/abs/2002.05709

[10] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," no. arXiv:2111.06377, Dec. 2021, arXiv:2111.06377. [Online]. Available: http://arxiv.org/abs/2111.06377

[11] Z. Xie, Z. Zhang, Y. Cao, Y. Lin, J. Bao, Z. Yao, Q. Dai, and H. Hu, "Simmim: A simple framework for masked image modeling," no. arXiv:2111.09886, Apr. 2022, arXiv:2111.09886. [Online]. Available: http://arxiv.org/abs/2111.09886

[12] Z. Qi, R. Dong, G. Fan, Z. Ge, X. Zhang, K. Ma, and L. Yi, "Contrast with reconstruct: Contrastive 3d representation learning guided by generative pretraining," no. arXiv:2302.02318, May 2023, arXiv:2302.02318. [Online]. Available: http://arxiv.org/abs/2302.02318

[13] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," no. arXiv:1511.06434, Jan. 2016, arXiv:1511.06434. [Online]. Available: http://arxiv.org/abs/1511.06434

[14] J. Zhou, C. Wei, H. Wang, W. Shen, C. Xie, A. Yuille, and T. Kong, "ibot: Image bert pre-training with online tokenizer," no. arXiv:2111.07832, Jan. 2022, arXiv:2111.07832. [Online]. Available: http://arxiv.org/abs/2111.07832

[15] Z. Huang, X. Jin, C. Lu, Q. Hou, M.-M. Cheng, D. Fu, X. Shen, and J. Feng, "Contrastive masked autoencoders are stronger vision learners," no. arXiv:2207.13532, Jan. 2024, arXiv:2207.13532. [Online]. Available: http://arxiv.org/abs/2207.13532

[16] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," no. arXiv:2103.03230, June 2021, arXiv:2103.03230. [Online]. Available: http://arxiv.org/abs/2103.03230

[17] C. Tao, X. Zhu, W. Su, G. Huang, B. Li, J. Zhou, Y. Qiao, X. Wang, and J. Dai, "Siamese image modeling for self-supervised vision representation learning," no. arXiv:2206.01204, Nov. 2022, arXiv:2206.01204. [Online]. Available: http://arxiv.org/abs/2206.01204

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," no. arXiv:1409.0575, Jan. 2015, arXiv:1409.0575. [Online]. Available: http://arxiv.org/abs/1409.0575

[19] T. T. Team. (2019, jan) Flowers. [Online]. Available: http://download.tensorflow.org/example_images/flower_photos.tgz

[20] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

Tackling Ambiguity in Visual Question Answering: A Clarification Feature with BLIP-2

Moe Ito
Mount Holyoke College
ito23m@mtholyoke.edu

Abstract— **This paper introduces a clarification feature to address the challenge of ambiguity in questions posed to Visual Question Answering (VQA) systems. Ambiguity often arises from vague pronouns (e.g., ”it,” ”this,” ”that”) or general terms (e.g., ”thing,” ”object”) without sufficient context. The proposed feature detects such ambiguities and prompts users to refine their questions, enhancing response accuracy. The implementation employs BLIP-2, a state-of-the-art vision-language model, and utilizes the VQA v2.0 dataset. A prototype system was developed and demonstrated with a specific image, successfully identifying and resolving ambiguous user questions. The results highlight the feasibility of incorporating a clarification mechanism to improve VQA systems. Future work will explore broader applications and more robust evaluation methods.**

Index Terms— **Visual Question Answering; Natural Language Processing; Ambiguity; Clarification; BLIP-2**

I. Topic Introduction

Visual Question Answering (VQA) is a task in computer vision that combines visual recognition and natural language processing. It involves generating a natural language answer to a question posed about a given image. This task is significant for its potential to bridge the gap between visual perception and language understanding, addressing both high-level reasoning and detailed visual comprehension.

VQA gained significant attention with the publication of the paper ”VQA: Visual Question Answering” by Antol et al. in 2015 [1], which introduced a standardized dataset and benchmark for the task, and with the subsequent competition held at CVPR 2016. Together, these developments established VQA as a prominent research area in computer vision, enabling researchers to evaluate models consistently and driving rapid advancements in the field. The potential applications of VQA are broad and impactful. It can assist visually impaired individuals by interpreting their surroundings, aid medical professionals in analyzing diagnostic images, and streamline tasks like content retrieval and safety inspections. For example, VQA systems could help identify hazards in industrial settings or enable users to locate specific scenes in videos efficiently.

To evaluate progress in VQA, datasets play a crucial role. One widely used benchmark is the VQA v2.0 dataset, proposed by Goyal et al. [2]. This dataset improves upon the original VQA dataset by addressing answer imbalance and doubling the number of image-question pairs to approximately 1.1 million. It pairs each question with two similar images that lead to different answers, ensuring robustness in model evaluation. For example, in Figure 1, the question ”Who is wearing glasses?” is associated with two images: one where a man is wearing glasses and another where a woman is wearing glasses. Both images include both a man and a woman, highlighting how the dataset carefully balances image-question-answer relationships. Figure 1 shows sample entries from the VQA v2.0 dataset, illustrating its open-ended question-and-answer pairs and its design to accommodate diverse visual contexts.

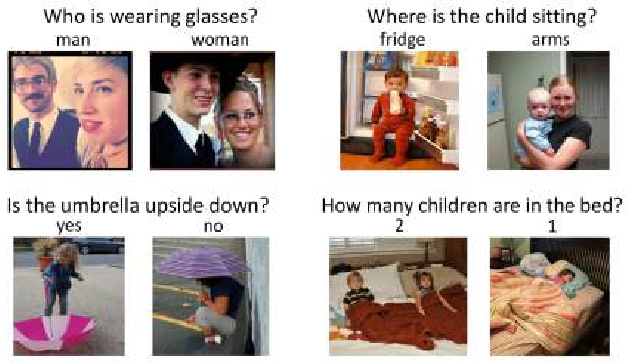

Who is wearing glasses?
man
woman
Where is the child sitting?
fridge
arms
Is the umbrella upside down?
yes
no
How many children are in the bed?
2
1

Fig. 1. Examples from the VQA v2.0 dataset.

II. Existing Methods

Traditional Approaches: Early Visual Question Answering (VQA) models relied on Convolutional Neural Networks (CNNs) for image feature extraction and Long Short-Term Memory networks (LSTMs) for processing question text. Antol et al. (2015) [1] demonstrated these methods on the original VQA dataset, using pre-trained VGGNet and simple question encoding techniques. While effective for straightforward queries, such as object recognition, these methods struggled with more complex questions requiring deeper reasoning, such as counting or spatial understanding. These limitations highlighted the need for more sophisticated mechanisms to align visual and textual information.

Attention Mechanisms and Transformers in VQA: Attention mechanisms laid the foundation for significant advancements in VQA. Lu et al. (2016) [3] introduced a Hierarchical Co-Attention Model that applied attention jointly to image regions and question words, capturing fine-grained alignments. This approach enhanced the alignment between visual and textual features, improving model accu-

TABLE I
SUMMARY OF EXISTING METHODS IN VQA

Method	Key Features	Strengths	Limitations
Traditional Approaches	CNNs for image features, LSTMs for question processing	Effective for simple VQA tasks	Struggles with complex queries
Co-Attention Models	Joint attention to image and question	Captures modality alignments	Computationally expensive
Transformer-Based Models	Self-attention mechanisms for vision and language	Accurate contextual understanding	High computational cost
BLIP	Pre-trained encoders for multimodal tasks	Versatile for VQA and captioning	Limited handling of ambiguous queries
BLIP-2	Combines frozen encoders with LLMs	High efficiency and scalability	Requires further tuning for ambiguity

racy on complex VQA tasks. Building on the success of attention mechanisms, the Transformer architecture introduced in ”Attention Is All You Need” (2017) revolutionized the field of machine learning. Transformer-based models such as ViLBERT [4] and UNITER [5] have significantly advanced VQA by enabling seamless integration of visual and textual inputs. These models utilize self-attention and co-attention mechanisms to align and interpret complex relationships within image-question pairs, setting new benchmarks in VQA accuracy.

BLIP and BLIP-2: BLIP and BLIP-2 are among the recently proposed models. BLIP (Bootstrapping Language-Image Pre-training), introduced by Salesforce in 2022 [6], serves as a unified framework for vision-language tasks such as image captioning and VQA. It leverages pre-trained encoders to align visual and textual modalities, enabling robust cross-modal reasoning. BLIP-2, a subsequent advancement proposed in 2023 [7], extends this framework by integrating frozen pre-trained vision models with large language models (LLMs). To bridge the gap between these two modalities, BLIP-2 employs a Querying Transformer (Q-Former), which facilitates efficient alignment while minimizing the number of trainable parameters. Despite its lightweight design, BLIP-2 achieves state-of-the-art performance across various vision-language tasks, demonstrating enhanced scalability and computational efficiency.

Table I summarizes key existing methods in VQA, highlighting their strengths and limitations. These methods demonstrate the evolution of techniques from traditional approaches to recent advances.

III. Open Challenges

A. Current Limitations

While existing VQA datasets and models assume that questions are clearly defined, real-world interactions often involve ambiguous questions with vague pronouns (e.g., ”it,” ”this”) or omitted context. This ambiguity hinders models from providing definitive answers, highlighting a key limitation in their ability to handle less structured, real-world queries [8].

In related fields like Knowledge-Based Question Answering (KBQA), mechanisms for generating clarification questions have been explored to address similar challenges [9]. However, such methods are still underexplored in VQA, where ambiguity in visual and textual contexts presents unique challenges. This gap underscores the need for novel approaches to detect and resolve ambiguities in VQA, enabling more practical and real-world applications.

B. Research Idea

To address the challenges inherent in existing VQA systems discussed in the previous section, this research aims to propose a VQA system equipped with a clarification feature. The following are the core components of this proposed idea:

1) **Ambiguity Detection**: The proposed system leverages BLIP-2’s pre-trained capabilities in both natural language and visual understanding to identify user queries that lack sufficient specificity. The system will automatically detect ambiguous expressions, such as vague pronouns (e.g., ”it,” ”this,” ”that”) and general terms (e.g., ”thing,” ”object”), by analyzing the interplay between the image context and the accompanying textual query. This enables the system to identify when a question requires additional clarification before an accurate answer can be generated.
2) **Clarification Question Generation**: Upon detecting ambiguity, the system generates follow-up questions to clarify the user’s intent. BLIP-2, fine-tuned with a custom dataset containing ambiguous queries and their corresponding clarification questions, is used to dynamically formulate questions that address the detected ambiguity. For example, if the user asks, ”What is it?” in the context of an image, the system might respond, ”Are you referring to the object on the sofa or the one near the table?” By leveraging BLIP-2’s ability to align visual and linguistic features, the system ensures that the generated questions are both context-aware and precise.

IV. Concept to Code

The runnable source code associated with this proposed research idea is available on GitHub and can be found here: `https://github.com/MHC-FA24-CS341CV/beyond-the-pixels-emerging-computer-`

`vision-research-topics-fa24/tree/main/code/14-vqa`.

A. Dataset Description

This program uses the VQA v2.0 dataset [2], which consists of:

- **Questions**: Open-ended questions about images that require an understanding of vision, language, and commonsense knowledge to answer.
- **Annotations**: Human-annotated answers to each question, including multiple acceptable responses.
- **Images**: Real-world images sourced from the MS COCO dataset, covering diverse scenes and objects.

This program specifically utilizes the validation set of the VQA v2.0 dataset, reflecting the project's focus on demonstrating inference capabilities of the pre-trained BLIP-2 model. The smaller size of the validation set further supports streamlined demonstrations, making it a practical choice for this purpose.

B. Run Instructions and Connection to Research Idea

To execute the program, follow these steps:

1) Access the GitHub folder for this program via the link provided at the start of this section.
2) Download the following three required files from the link detailed in the README.md:
   - `v2_OpenEnded_mscoco_val2014_questions.json`
   - `v2_mscoco_val2014_annotations.json`
   - `val2014.zip`
3) Upload these files to your Google Drive.
4) Open the .ipynb file in Google Colab by clicking the **"Open in Colab"** button displayed at the top of the file.
5) Modify the file paths in the notebook to reflect the location of the dataset files in your Google Drive.
6) Run each cell in the notebook sequentially to observe the system's functionality and interact with the clarification feature.

This program connects directly to the proposed research idea by demonstrating the clarification feature in action. Specifically, it shows how ambiguous user queries are detected and resolved interactively, aligning with the research objective of enhancing VQA systems' contextual understanding and accuracy.

C. Results

To demonstrate the functionality of the system, a specific image from the validation set of the VQA v2.0 dataset was used. Figure 2 shows the image we used, along with the interactive VQA process shown in Figure 3. This process demonstrates how the system identifies ambiguous questions, prompts the user for clarification, and generates a final answer.

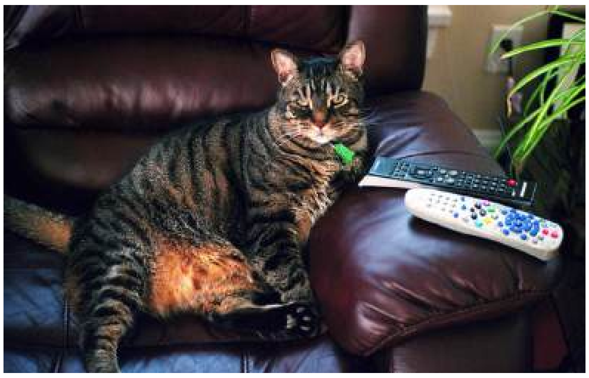
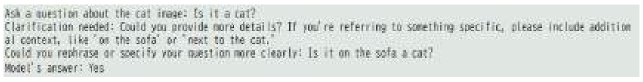


Fig. 2. The cat image used in the program.



```
Ask a question about the cat image: Is it a cat?
Clarification needed: Could you provide more details? If you're referring to something specific, please include addition
al context, like 'on the sofa' or 'next to the cat.'
Could you rephrase or specify your question more clearly: Is it on the sofa a cat?
Model's answer: Yes
```

Fig. 3. Output example of the interactive VQA process with clarification feature.

D. Future Work and Technical Implementations

The current implementation serves as a simplified demonstration, focusing on a specific image and predefined ambiguous terms to test the concept of ambiguity detection and clarification in VQA. However, scaling this idea to a robust and user-friendly system capable of handling diverse images and user-generated questions involves addressing several technical challenges:

- **Automated Ambiguity Detection**: The current implementation relies on manually defined ambiguous terms and context phrases. Future work involves fine-tuning BLIP-2 on a custom dataset containing diverse images and questions labeled with ambiguity annotations, enabling automatic detection of vague queries.
- **Scalability to Diverse Inputs**: While the current implementation is limited to a single image, the ideal system would allow users to upload arbitrary images and ask open-ended questions. Extending the system to dynamically handle diverse visual and textual contexts is crucial for broader applicability.
- **Dataset Expansion**: A large-scale dataset that includes ambiguous questions, clarified versions, and corresponding answers across various scenes and objects is essential. Such a dataset would facilitate fine-tuning and improve the system's ability to generalize to real-world scenarios.

These advancements aim to enhance the scalability and robustness of the system, enabling it to handle diverse images and questions effectively while maintaining high interpretive accuracy.

REFERENCES

[1] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "Vqa: Visual question answering," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2425–2433.

[2] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, "Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] J. Lu, J. Yang, D. Batra, and D. Parikh, "Hierarchical question-image co-attention for visual question answering," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/9dcb88e0137649590b755372b040afad-Paper.pdf

[4] J. Lu, D. Batra, D. Parikh, and S. Lee, "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks," *Advances in neural information processing systems*, vol. 32, 2019.

[5] Y.-C. Chen, L. Li, L. Yu, A. El Kholy, F. Ahmed, Z. Gan, Y. Cheng, and J. Liu, "Uniter: Universal image-text representation learning," in *European conference on computer vision*. Springer, 2020, pp. 104–120.

[6] J. Li, D. Li, C. Xiong, and S. Hoi, "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *International conference on machine learning*. PMLR, 2022, pp. 12 888–12 900.

[7] J. Li, D. Li, S. Savarese, and S. Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," in *International conference on machine learning*. PMLR, 2023, pp. 19 730–19 742.

[8] S. Kottur, J. M. Moura, D. Parikh, D. Batra, and M. Rohrbach, "Visual coreference resolution in visual dialog using neural module networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 153–169.

[9] J. Xu, Y. Wang, D. Tang, N. Duan, P. Yang, Q. Zeng, M. Zhou, and X. Sun, "Asking clarification questions in knowledge-based question answering," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 1618–1629.