A7 Image Analysis Coursework Report

jn492

June 29, 2025

Word count: 2924

1 Introduction

This report is divided into three sections, each corresponding to one module: classical image analysis, image reconstruction with machine learning techniques, and image quality evaluation.

2 Module 1

This section explores classical image processing techniques to classify butterfly images. The dataset is displayed in Figure 1.

Figure 1: Visualisation of dataset

2.1 Color Classification

This subsection outlines the pipeline of how butterflies were automatically classified into color groups. Firstly, the images were first cropped to increase the butterfly-to-background ratio, as shown in Figure 2.

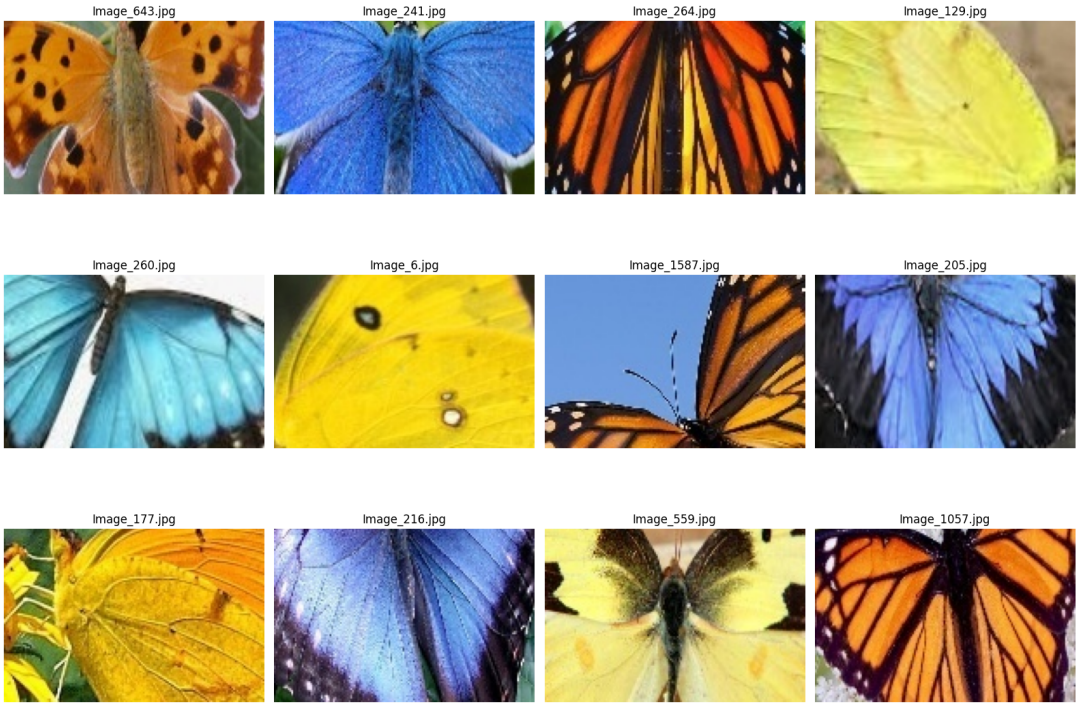


Figure 2: Cropped images

Image 1587 shows an orange butterfly against a smooth blue background, often misclassified as a blue butterfly due to background influence. To reduce this effect, a mask was applied by computing HSV brightness over 5×5 patches, thus removing areas with low variation. Note that classical background removal techniques is not used here as it will be addressed in Section 2.2. As shown in Figure 3, this effectively removes the uniform backgrounds. However, it may also remove dark butterfly regions, such as wing edges as seen in image 264. However, this trade-off is acceptable since the focus is on wing color for this task. Though this method may fail for dark butterflies on cluttered backgrounds.

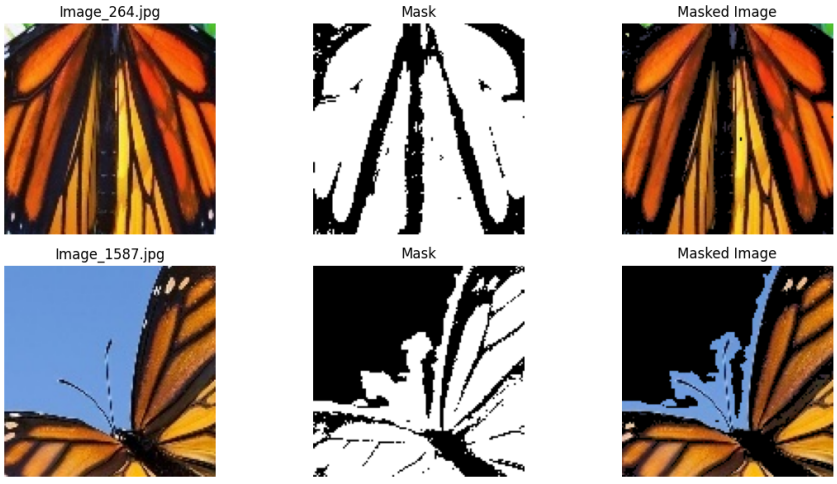


Figure 3: Removing areas with low variation

After masking, median HSV values were computed per image as a classification metric. HSV was chosen over RGB for its interpretability, as hue captures color, saturation reflects vibrancy, and value indicates brightness. Medians were used instead of means to reduce outlier influence and avoid misleading color blending. When plotting the HSV channels in Figure 4, there is noticeable overlap between yellow and orange classes due to similar hue and brightness.

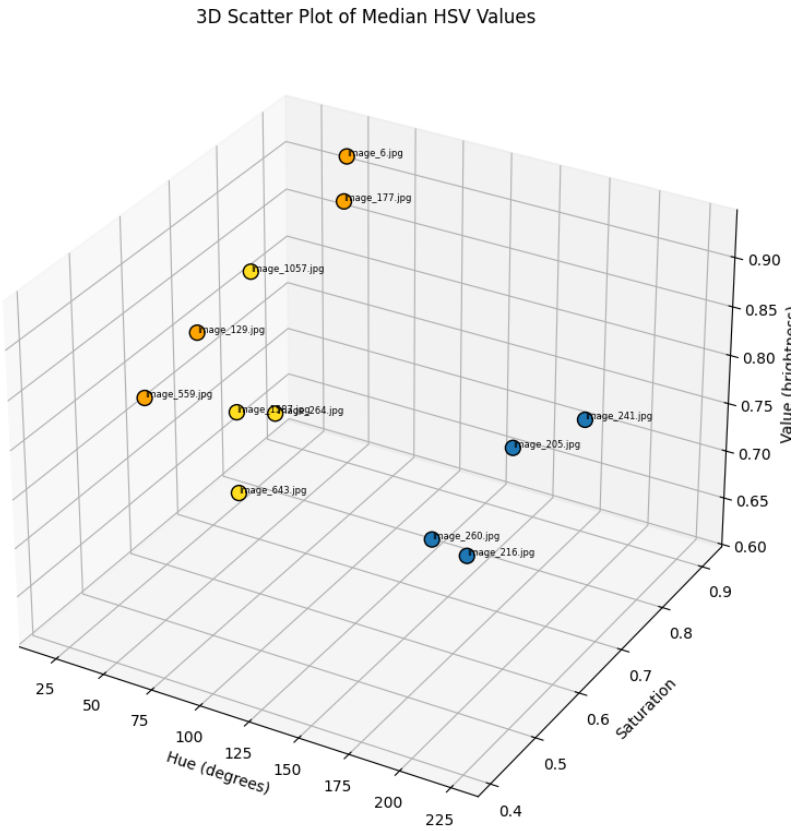


Figure 4: Median Hue, Saturation and Value for each image

To better separate similar colors like yellow and orange, the Euclidean distance in HSV space was weighted to emphasize the value (brightness) channel. Image pairs with distances below a manually set threshold of 35 were grouped together. Classification results are shown in Figure 5, with each row representing a distinct color group.

Figure 5: Classification results (grouped by row)

2.2 Background removal

The watershed algorithm was chosen for its simplicity and effectiveness in gradient-based segmentation, as butterfly edges are typically well-defined against the background. The image was converted to grayscale, smoothed with a Gaussian filter, and gradients were computed using a Sobel filter. A small central rectangle was used as the foreground marker, and the image borders as background markers, guiding the region-growing process. This produced the mask in Figure 6 and the segmentation in Figure 7.

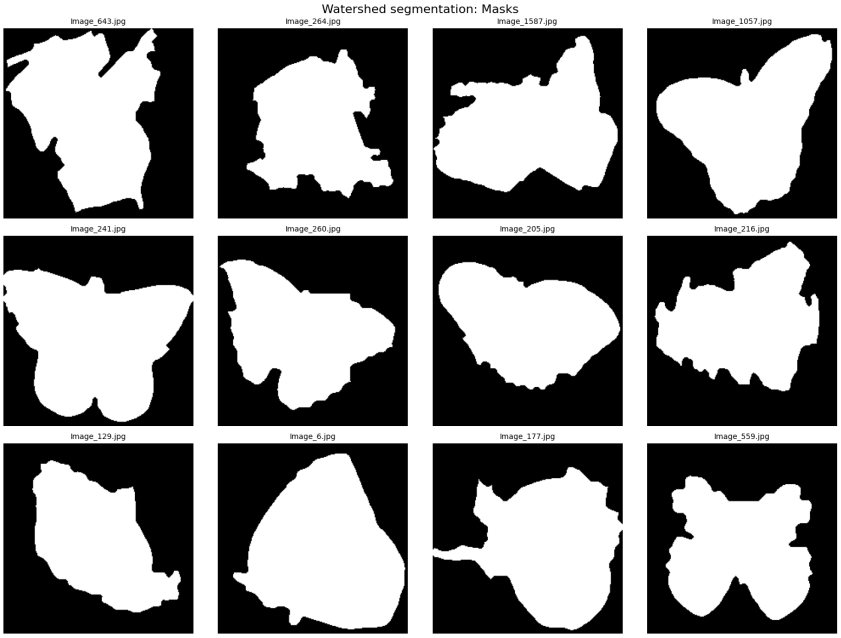


Figure 6: Watershed segmentation masks

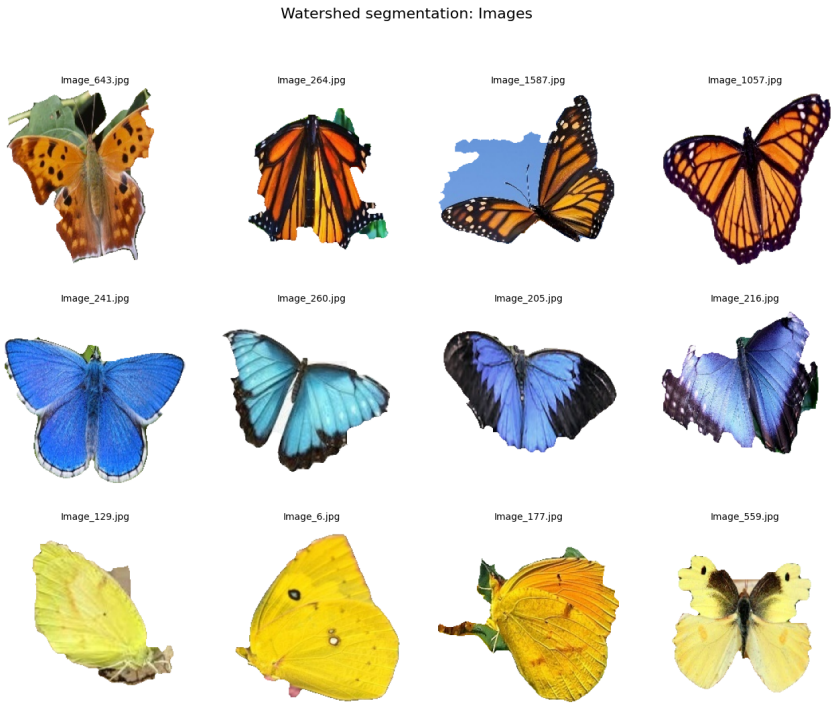


Figure 7: Watershed segmentation

This method performed well in cases with strong contrast between the butterfly and background. However, it struggled when (1) the butterfly's edges lacked clear contrast such as in image 0216 and (2)

the butterfly was not centered, as in image 1587, leading to background regions being misclassified as foreground.

Alternatively, the GrabCut algorithm was tested, as it adapts to the color distribution of each image, which may improve segmenting these butterfly images with varying backgrounds and foreground colors [6]. GrabCut begins with an initial rectangle assumed to enclose the foreground. It models foreground and background pixel colors using Gaussian Mixture Models (GMMs) and iteratively refines the segmentation to maximize the likelihood of observed RGB values. Though there is some debate whether Grabcut is a classical image analysis technique, as it does require user input for each image, and does not generalise learnt parameters for any image, this report deems it a classical statistical technique and thus suitable for this classical image processing task. The results are shown in Figure 8.

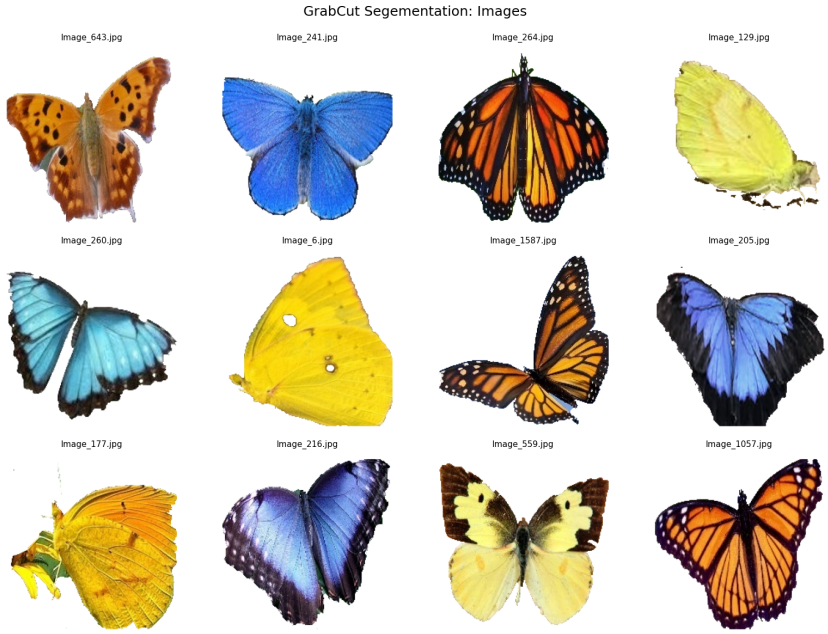


Figure 8: Grabcut background removal of butterflies

2.3 Collection display

A function was developed to display selected butterfly images for visual inspection or presentation. It accepts a dictionary of images, the number of butterflies to show, background RGB color, image cell size, and display resolution. Figure 9 shows an example output with two butterflies displayed on the default light purple background (RGB: 230, 220, 255).

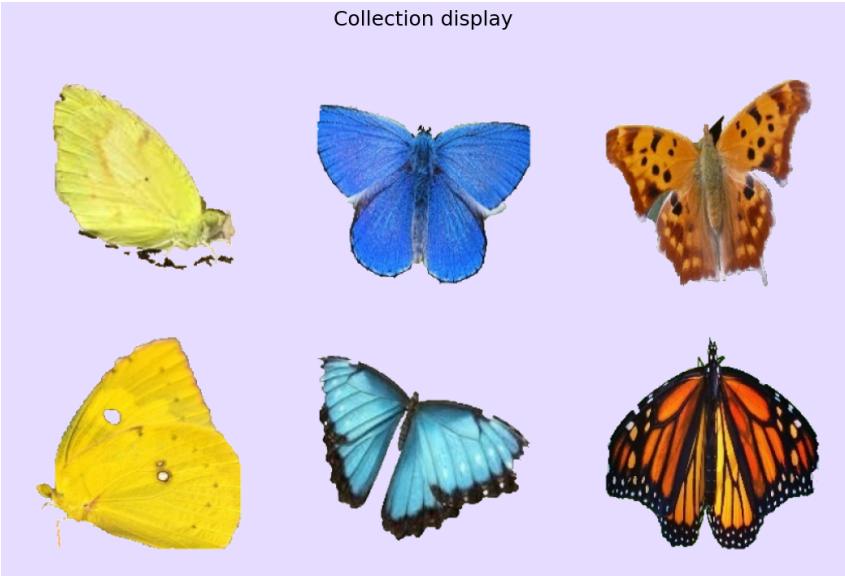


Figure 9: Collection

2.4 Identify the Outlier

The task is to identify the butterfly that differs most from a group of similarly colored specimens. A key visual difference is the presence or absence of a dark rim around the wings. For this reason, brightness in the HSV color space (Value channel) was selected as an initial proxy.

Mean brightness was computed for each image, and pairwise Euclidean distances were calculated within each color group. The butterfly with the highest total distance was labeled as the outlier. However, this misclassified image 264 as the outlier among orange butterflies as seen in Figure 10. Despite its dark rim, image 264 is overall much darker than others, inflating its distance values compared to the true outlier, image 643, which lacks a rim.

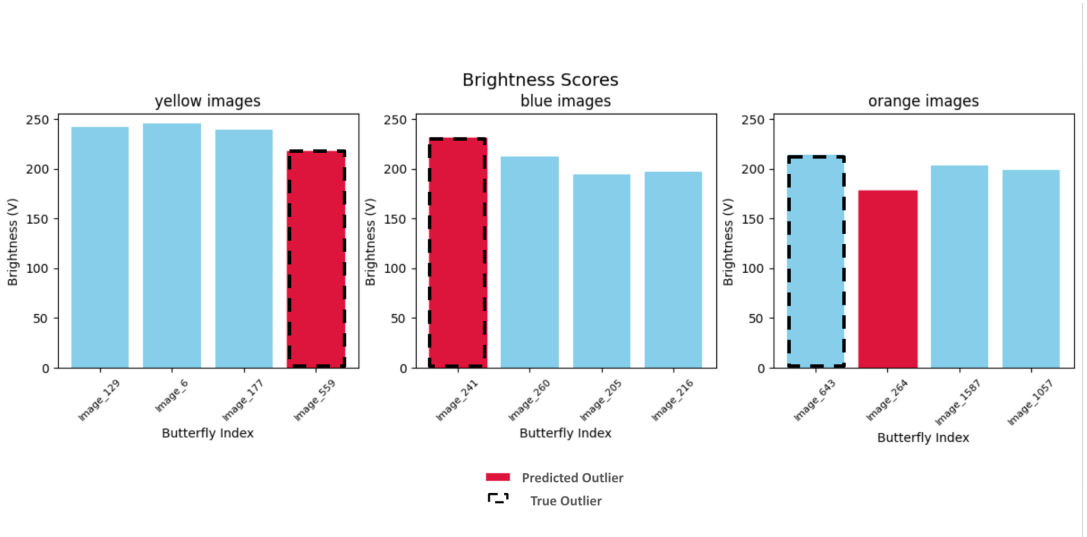


Figure 10: Using image brightness values (blue) euclidean distances to predict outliers (red). The true outlier (dotted) was not correctly predicted for image 643.

To improve performance, the Canny edge detector was applied to isolate wing boundaries, where dark rims typically appear. Within these edge regions, the average pixel darkness was calculated. The butterfly with the greatest deviation from the group mean was then identified as the outlier. This method improves upon brightness-only analysis by targeting structural features relevant to species identification, particularly the presence or absence of a dark rim. Results are shown in Figure 11.

Figure 11: Using edge detection to predict odd one out

3 Module 2: Machine Learning for Denoising and Inpainting

This section examines image quality enhancement using the Plug-and-Play Alternating Direction Method of Multipliers (PnP-ADMM) framework for denoising and inpainting.

3.1 Denoising of blurred images

Image reconstruction is formulated as a linear inverse problem, recovering a clean image from noisy or incomplete data by balancing data fidelity and prior knowledge. The PnP-ADMM algorithm alternates updates of three variables: x (data fidelity), v (denoised prior), and u (dual variable enforcing consistency). Traditional regularizers are replaced by a pretrained U-Net denoiser to better capture natural image structure. Image 0006 is used as the ground truth in this section. The implementation is shown in Figure 12, where the x-update is computed via the conjugate gradient method and the v-update applies the denoiser.

```python
def pnp_admm(y,eta,A,AT,tolerance,iter,model):
    """
    Plug-and-Play ADMM for image reconstruction using a learned denoiser.

    This function solves an inverse problem using the Plug-and-Play Alternating Direction
    Method of Multipliers (PnP-ADMM) framework. It alternates between reconstructing an
    image to match measured data and denoising it using a pretrained model.

    Parameters:

        y (torch.Tensor): Observed measurement (e.g., blurred or masked image).

        eta (float): Regularization parameter that balances data fidelity and prior.

        A (function): Forward operator that simulates the degradation (e.g., blur or mask).

        AT (function): Adjoint (transpose) of the forward operator A.

        tolerance (float): Tolerance for the conjugate gradient solver.

        iter (int) :  Number of ADMM iterations.

        model (torch.nn.Module): Pretrained denoising model (e.g., DnCNN or U-Net).

    Returns:
        reconstructed image (torch.Tensor): Reconstructed image after ADMM iterations, clamped to [0, 1].

    """
### 1.  Initialise the variable:

    #This is the best guess of the image (filled with zeros, with the blurred image dimensions)
    x = torch.zeros_like(y)

    # v is a denosied version of x + u, dynamically updated each iteration.
    v = torch.zeros_like(y)

    # This is the dual variable. This tracks the difference between the reconstructed image x and the denoised image v.
    u = torch.zeros_like(y)

### 2. Calculate the x update:

    # first part of the x-update
    def C(x):
        return AT(A(x)) + eta * x

    for iteration in tqdm(range(iter),desc="Iteration number:"):
        # second part of the x-update
        b = AT(y) + eta * (v - u)

        # Solve x in Cx = b using the conjugate gradient
        x = conjugate_gradient(C, b, x, iter, tolerance)

### 3. Calculate the v update (make a denoised reconstruction)
        with torch.no_grad():
            denoiser_input = torch.clamp(x + u, 0, 1)
            v = model(denoiser_input)
### 4. Calculate the u update (make the denoised reconstruction, and data-driven reconstruction match)
        u = u + (x - v)
    return torch.clamp(x, 0.0, 1.0)
```

Figure 12: PnP-ADMM function

To simulate degradation, image 0006 was blurred using averaging kernels of size 7×7, 13×13, and 17×17. Each blurred image was reconstructed using `PnP-ADMM` over 20 iterations with regularization and tolerance set to 10^{-4}. As shown in Figure 13, MSE improves by a whole magnitude from $k = 7$ to $k = 13$, and doubles when increasing to $k = 17$. Visible square artifacts likely result from the pretrained denoiser, which may have been trained on small patches and struggles on larger images.

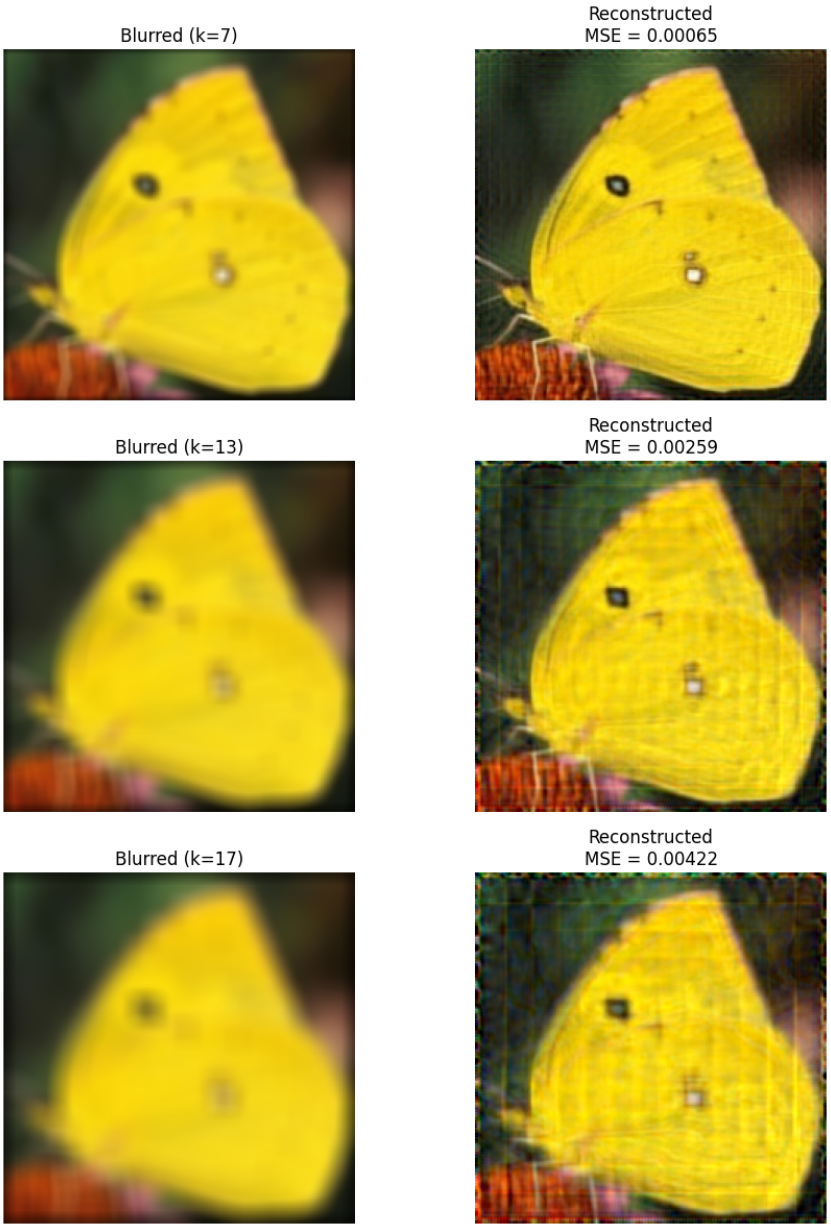


Figure 13: Deblurring results using the `pnp-admm` function.

3.2 Denoising with of blurred images with gaussian noise

Using a blur kernel size of $k = 13$, Gaussian noise with zero mean and a standard deviation of 0.01 was added to the blurred image. The `PnP-ADMM` algorithm was then run again to reconstruct the image, and the results are shown in Figure 14. The MSE for the noisy blurred image is approximately twofold higher than the blurred image without added Gaussian noise. The reconstructed image appears noticeably grainier, as gaussian noise introduces random high-frequency perturbations that is difficult to resolve during reconstruction.

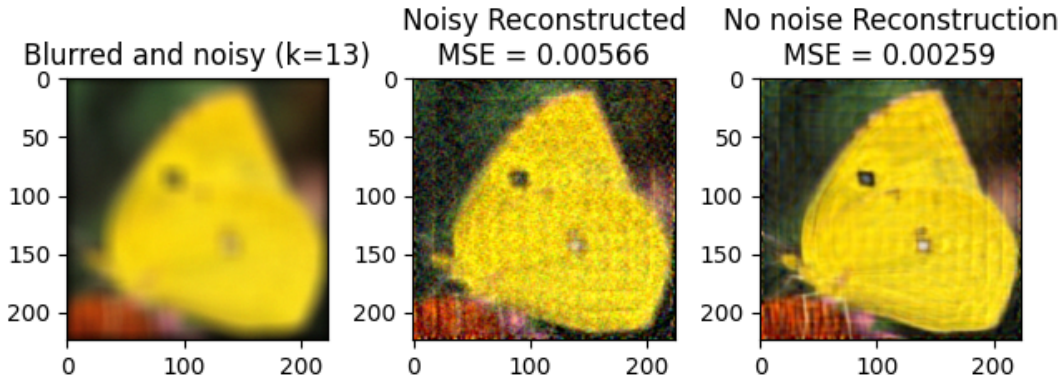


Figure 14: Reconstruction results using `pnp-admm` on a blurred image with added Gaussian noise.

3.3 Image Inpainting to Recover Missing Pixels

This subsection addresses image inpainting, aiming to reconstruct images with missing pixels. To simulate this, 40%, 60%, and 80% of pixels in image 0006 were randomly removed. Reconstruction using PnP-ADMM was performed over 20 iterations with a tolerance of 10^{-4} and regularization parameter set to 1. Results are shown in Figure 15.

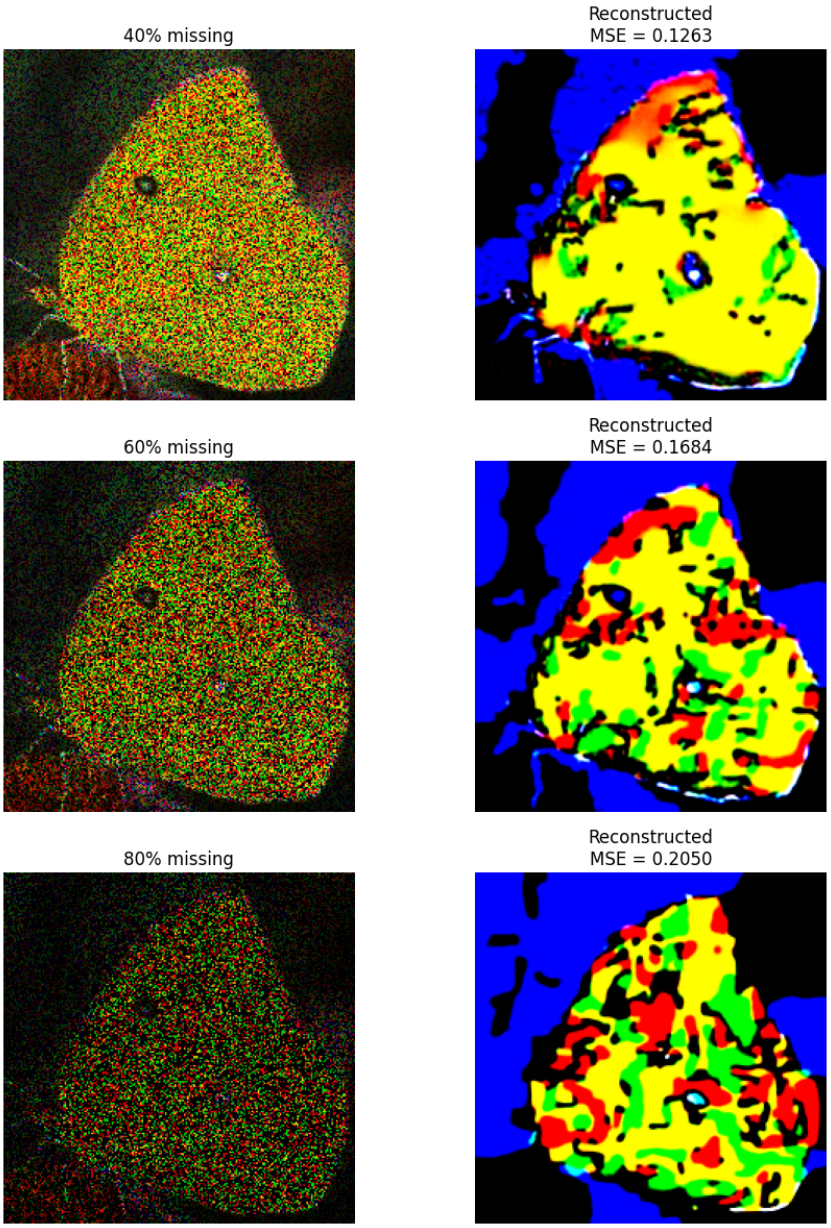


Figure 15: PnP ADMM inpainting results with 40%, 60% and 80% of pixels missing.

None of the reconstructions exhibit a natural appearance, with notable loss of detail, texture, and colour across all levels of missing pixels. At 40% missing data, some structural smoothing is observed, especially near the wing, suggesting that the denoiser can partially recover natural features. However, as the proportion of missing pixels increases, the MSE rises and the reconstructions degrade further, losing coherence. This poor performance indicates that PnP-ADMM struggles with inpainting under severe degradation, likely due to insufficient data in the original image to constrain the solution. Additionally, Nair et al. [5] note that CNN denoisers often violate the nonexpansive property, meaning they can amplify image differences rather than suppress them, leading to divergence in PnP-ADMM. Since the U-Net used in this project is a CNN-based denoiser, it likely contributed to the generation of spurious reconstructions.

3.4 Image Inpainting with PnP-RED

Another variant of the plug-and-play framework is Regularization-by-Denoising (RED), referred here as PnP-RED. This approach defines the regularizer with a pretrained denoiser, and reconstruction is performed via gradient descent. A regularization strength of 0.1 and learning rate of 1 are used. The implementation is shown in Figure 16.

```python
def pnp_red(y, A, AT, iter, model, step_size=0.1, eta=1.0):
    """
    Implements the PnP-RED gradient descent algorithm.

    Args:
        y: Corrupted image (with missing pixels).
        A: Forward operator.
        AT: Transpose of A.
        iter: Number of iterations.
        model: Denoising model (D(x)).
        step_size: Regularisation strength.
        eta: Gradient desscent Step size.

    Returns:
        Reconstructed image x.
    """
    # Initialize x (e.g., zero image with same shape as y)
    x = torch.zeros_like(y)

    for i in tqdm(range(iter),desc="iteration"):
        Ax = A(x)
        residual = AT(Ax - y)  # gradient of data term
        denoised = model(x)    # D(x)
        grad_J = residual + step_size * (x - denoised)

        # Gradient descent step
        x = x - eta * grad_J
    return torch.clamp(x, 0.0, 1.0)
```

Figure 16: PnP-RED function

Figure 17 compares PnP-RED and PnP-ADMM reconstructions with 60% of pixels removed. PnP-RED produces more structural details and natural textures, with a smaller MSE. This is likely due to the absence of an explicit objective function for PnP-ADMM, which can lead the algorithm to diverge and create spurious reconstructions [3]. In contrast, PnP-RED tends to converge more reliably because it incorporates a well-constrained, minimizable formulation of the regularizer.

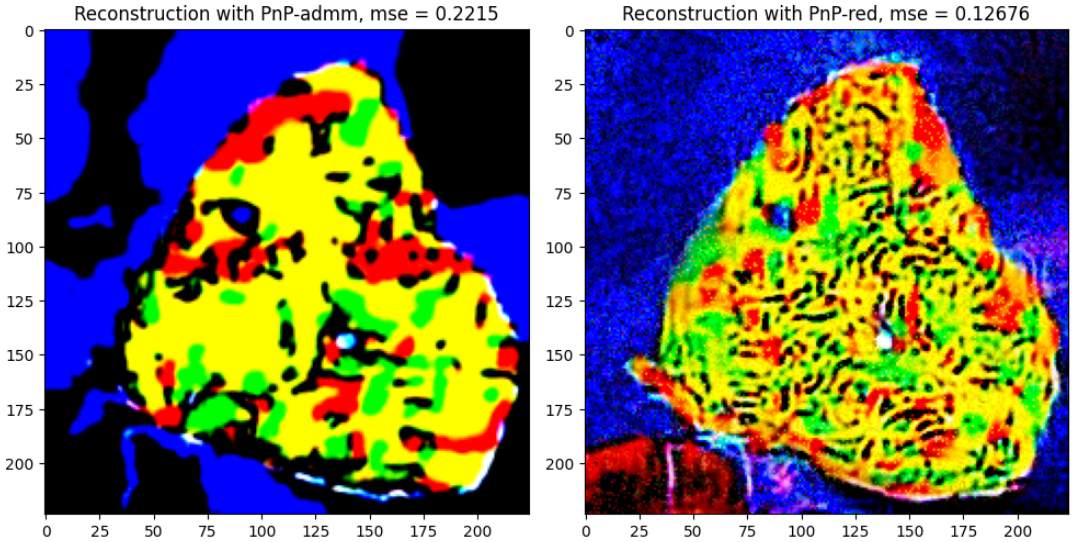


Figure 17: impainting pnp-red

Note that the regularizer gradient is approximated as $\nabla\rho(x) = x - D(x)$, where $D(x)$ is the denoised image. This approximation is valid under two conditions: (1) the denoiser is Jacobian symmetric, meaning pixel influence is bidirectionally equal, and (2) it is locally homogeneous, responding linearly to small intensity changes. When these conditions hold, the denoiser defines a valid regularization function, allowing the gradient to be efficiently computed via simple pixel-wise subtraction, without explicitly evaluating the Jacobian.

3.5 Effect of the number of iterations on deblurring and inpainting

This section examines how the number of iterations affects the performance of deblurring and inpainting, evaluated using the mean squared error relative to the ground truth image. Two blur kernel sizes (7 and 17) and two levels of missing pixels (40% and 80%) are selected for this analysis. The results are presented in Figure 18 and Figure 19.

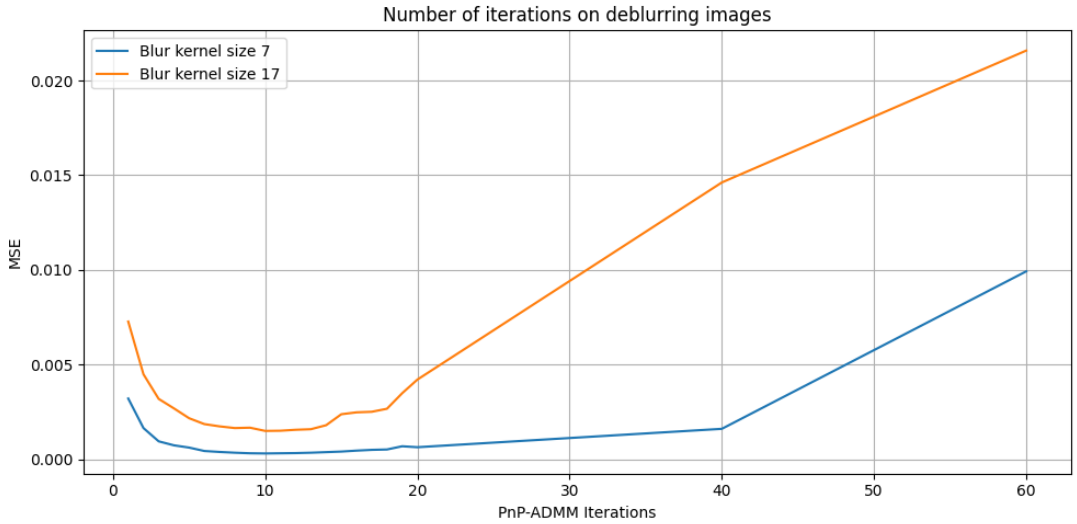


Figure 18: Deblurring with PnP-ADMM using kernel 7 and 17 with increasing iterations

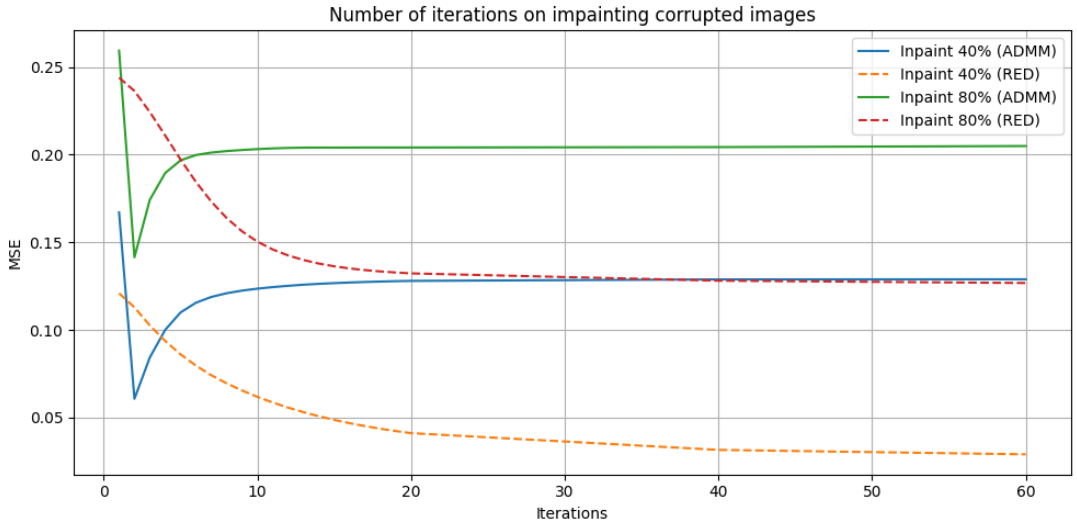


Figure 19: Impainting with PnP-ADMM and PnP-RED using missing pixels 40% and 80% with increasing iterations

In the deblurring task, the PnP-ADMM algorithm reached its lowest MSE at approximately 10 iterations. Beyond this point, performance declined, with MSE increasing in later iterations. Among the tested blur kernels, the 7×7 kernel consistently yielded lower MSE than the 17×17 kernel, which is expected since the image blurred with the smaller kernel is more similar to the original and therefore easier to reconstruct.

In the inpainting task, the behavior of the algorithms diverges notably. For PnP-ADMM, the MSE initially decreases, reaching a minimum after approximately three iterations, but then begins to rise and stabilizes around eight iterations. This pattern aligns with the observations of Athalye et al. [1], who note that PnP-ADMM lacks a convergence guarantee, and thus more iterations may lead to divergence,

ultimately degrading reconstruction quality. As expected, more severely degraded images such as those with 80% missing pixels result in higher MSE compared to cases with less corruption (e.g., 40%). In contrast, PnP-RED exhibits stable and consistent improvement, with the MSE steadily decreasing and converging to a lower value. This suggests that PnP-RED offers more stable convergence than PnP-ADMM.

To mitigate the instability observed in PnP-ADMM, early stopping may be employed when MSE begins to rise. However, switching to PnP-RED may be a more robust alternative due to its consistent convergence behavior.

4 Module 3

4.1 Limitations of PSNR and SSIM for Evaluating Image Quality

Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) are widely used full-reference image quality assessment (FR-IQA) metrics. When applying PSNR and SSIM to the deblurring and inpainting reconstructions from module 2, Figure 20 shows that both metrics have limited ability to reflect perceptual image quality. PSNR and SSIM remains nearly unchanged across different reconstructions, even when inpainting reconstructions appear visually worse than deblurred images. For example, image h (inpainting) and image a (deblurring) receive similar PSNR and SSIM scores, despite image h having significant structural loss.

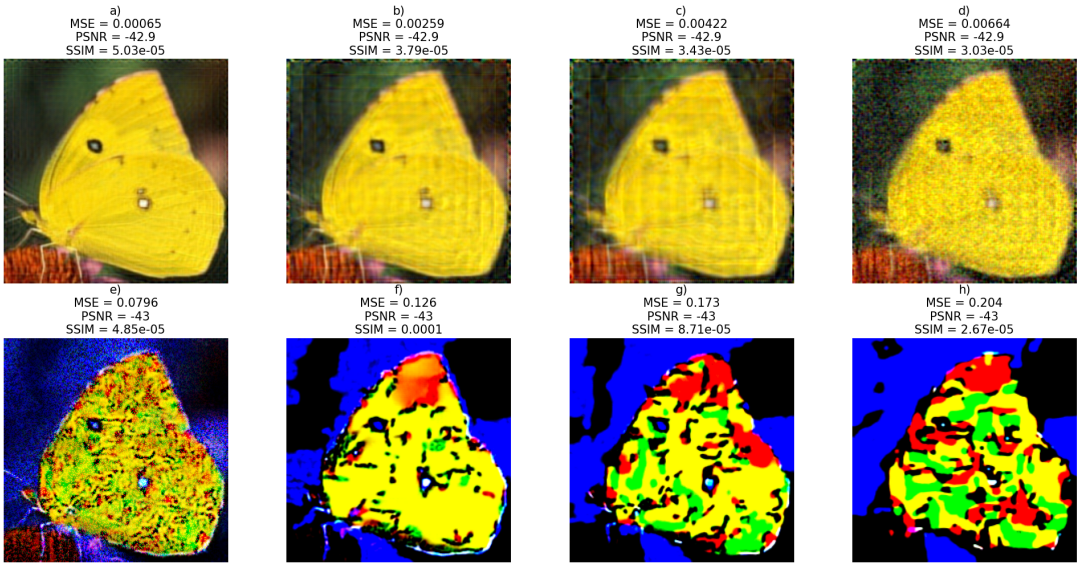


Figure 20: PSNR and SSIM evaluation on Module 2 reconstructions

This limitation arises because PSNR is sensitive to global pixel differences as it is a scaled MSE metric, whereas SSIM averages over patches and fails to detect large, spatially coherent degradations. As a result, both metrics can be misleading when evaluating images with missing textures or object parts [2].

More suitable metrics for this task are metrics that will reward images with good visual realism, structural completeness, and edge preservation. For FR-IQA, the Learned Perceptual Image Patch Similarity (LPIPS) metric may be a good alternative, as LPIPS computes distances between deep features extracted from neural networks, capturing more global structures. In addition, Perceptual Image Quality Evaluator (PIQE) could be a better NR-IQA, as it analyses high spatially active blocks to evaluate distortion and artefacts [4]. When applied, figure 21 shows these metrics perform better than PSNR and SSIM in quantifying perceptual quality, where LPIPS and PIQUE both penalise the poor inpainting reconstructions more than the deblurred reconstructions.

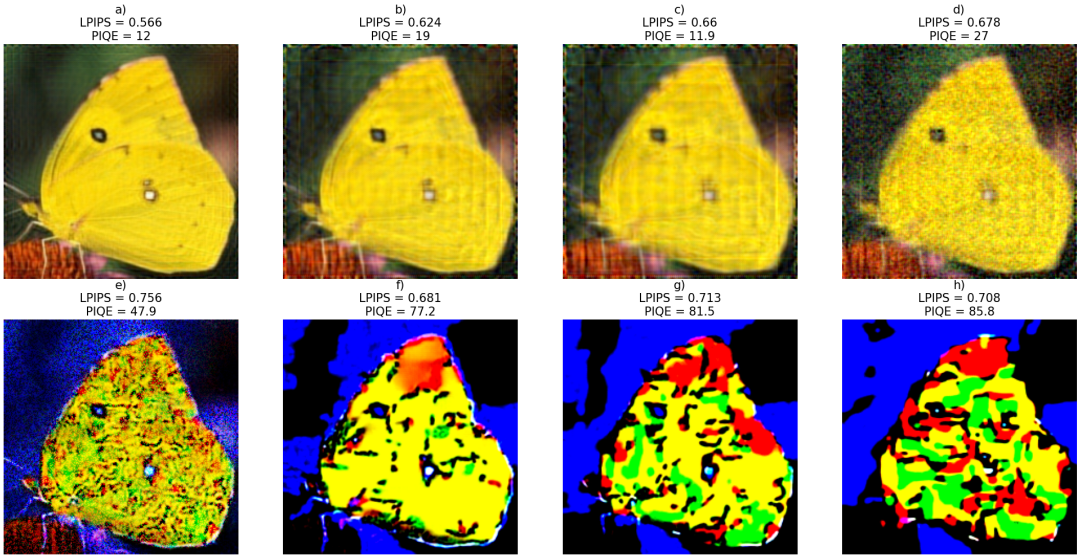


Figure 21: LPIPS and PIQE evaluation

4.2 PSNR and SSIM on degraded images with and without background removal

Five distortions were applied to a butterfly image 0216 to evaluate the behavior of PSNR and SSIM under different types of degradation: Gaussian noise, contrast enhancement, luminance reduction, JPEG compression, and missing region. Despite large visual differences, PSNR and SSIM scores did not change much, with PSNR values between 13.23 and 24.63, and SSIM scores between 0.550 and 0.924. For example, the image with a missing region scored similarly to a darkened image, as shown in Figure 22.

Figure 22: PSNR and SSIM on degraded images (with background)

While PSNR ranked JPEG compression as the best reconstruction, SSIM favored the image with a local patch missing. This aligns with the paper by Hore et al., where PSNR is less sensitive to JPEG compression than SSIM [2]. This is likely because PSNR computes pixel-wise differences, thus is sensitive to small scale changes but is tolerant to large scale compression artefacts. On the other hand, SSIM considers local structure, thus is more sensitive to global changes like compression noise.

Evaluations with the background removed using GrabCut (Figure 23) show increased PSNR and SSIM values, indicating that background pixels introduced error into the original calculations. However, the relative ranking of degradations remained unchanged, such that PSNR still favored JPEG compression, and SSIM preferred the image with a missing patch. This suggests the background contributed a consistent bias across all distortions without affecting metric sensitivity to degradation type.

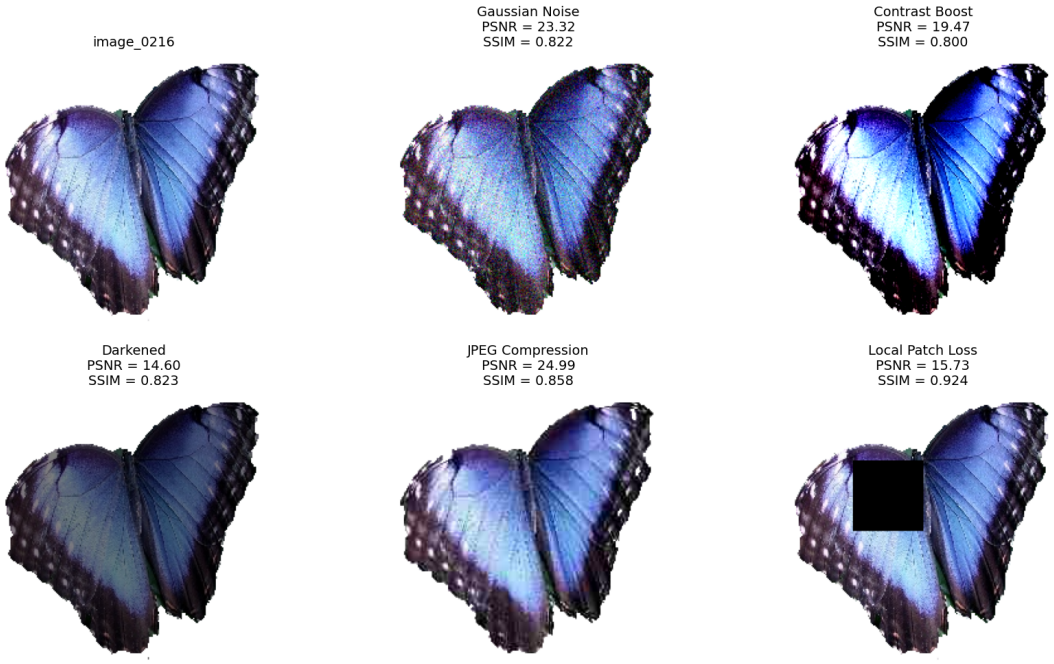
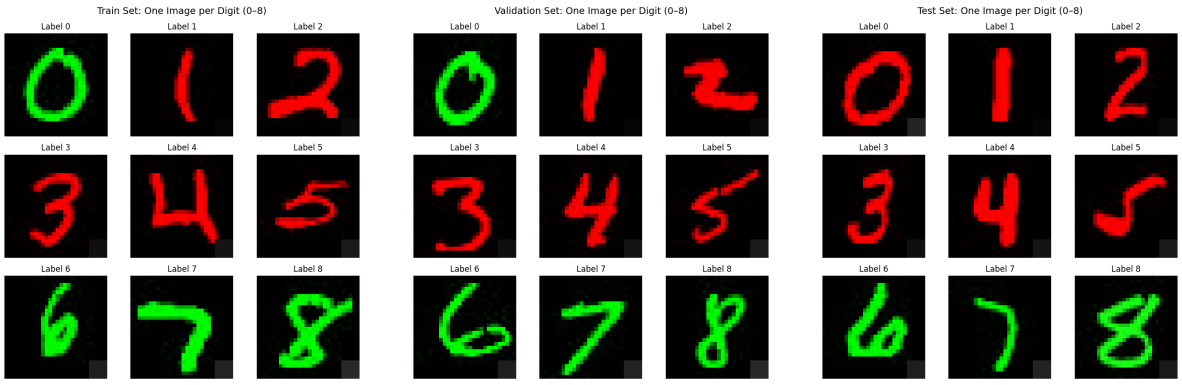


Figure 23: PSNR and SSIM on degraded images with background removed

4.3 MNIST classification with case A

To understand the challenges of machine learning in image analysis, this section explores the classification of the MNIST dataset using two different architectures. When `case == 'a'`, the model is a fully connected neural network with three layers, using Tanh activations and a final Softmax layer to output class probabilities for ten digits. The model is trained using the Adam optimizer with a learning rate of 0.001.

The train, validation and test data sets consists of colored images of MNIST digits, ranging from 0 to 9. Thehe first sample from each class for each set is shown in Figures 24a, 24b, and 24c.



(a) Train images (b) Validation images (c) Test images

Figure 24: Dataset images comparison

Common pitfalls are addressed through multiple strategies. Early stopping prevents overfitting by

monitoring validation loss and halting training when no improvement is observed over ten epochs. Image data is normalized to a range of 0 to 1, which stabilizes training and prevents gradient explosion. The training data is shuffled before each epoch, helping to reduce ordering bias since the dataset is split into training and validation sets using even and odd indices, respectively.

Bias may potentially be introduced through sampling bias for the even-odd splitting method of train and validation images, if there is data ordering present.

Data partitioning is performed by loading `0_development_data.pkl`, with even-indexed images assigned to the training set and odd-indexed to the validation set. The test set is loaded separately from `0_test_data.pkl`. However, there are color inconsistencies across splits. Generally, digits 1 to 5 are red, 6 to 8 are green for all train, validation and test sets. However, class 0 appears exclusively green in training/validation and exclusivey red in testing.

Model performance is demonstrated in Table 1, showing strong results for most classes, except for classes 0 and 5. Class 0 has zero precision and recall, likely due to the color mismatch, which caused the model to associate the digit with the training green color and fail to generalize to a red digit. Class 5 shows high recall but low precision, suggesting frequent misclassification of other digits like 0 to class 5, likely due to shared red color and similar curved edge of the digit. As false and true positives seem to be the main issue with these MNIST classifications, precision is the most appropriate metric to evaluate this dataset.

Table 1: Classification results for case A, rounded to 3 decimal places

Class	Accuracy	Sensitivity (Recall)	Specificity	Precision
0	0.901	0.000	0.999	0.000
1	0.998	0.991	0.999	0.995
2	0.996	0.973	0.998	0.983
3	0.995	0.978	0.997	0.976
4	0.996	0.996	0.996	0.967
5	0.902	0.991	0.893	0.479
6	0.997	0.989	0.998	0.979
7	0.996	0.972	0.998	0.987
8	0.994	0.969	0.997	0.973
9	0.996	0.980	0.997	0.977

Improvements include merging the training, validation and test datasets, followed by a randomized split to create new training, validation, and test sets. This ensures consistent feature distributions across splits and helps reduce color-based overfitting.

4.3.1 Implementing suggested improvements

The improvements were implemented by merging the data files, then randomly shuffling and splitting them using `train_test_split()` from `sklearn.model_selection` into a 75:15:15 train:validation:test ratio. As shown in Table 2, the model achieved clear improvements with near-zero loss across most classes.

Table 2: Improved classification results for case A, rounded to 3 decimal places

Class	Accuracy	Sensitivity (Recall)	Specificity	Precision
0	0.999	0.995	1.000	1.000
1	0.999	0.989	1.000	1.000
2	0.998	0.993	0.998	0.983
3	0.998	0.991	0.998	0.986
4	0.999	0.998	0.999	0.995
5	0.986	0.998	0.978	0.729
6	0.998	0.997	0.998	0.982
7	0.997	0.989	0.999	0.991
8	0.996	0.988	0.999	0.987
9	0.997	0.992	0.999	0.983

However, precision issues remain for digits 5 and 8, likely due to class imbalance, as shown in Figure 25. These classes have the fewest training examples, limiting the model's ability to generalize and making them harder to distinguish from others. A potential solution is to balance the dataset by subsampling all classes to match the size of the smallest class.

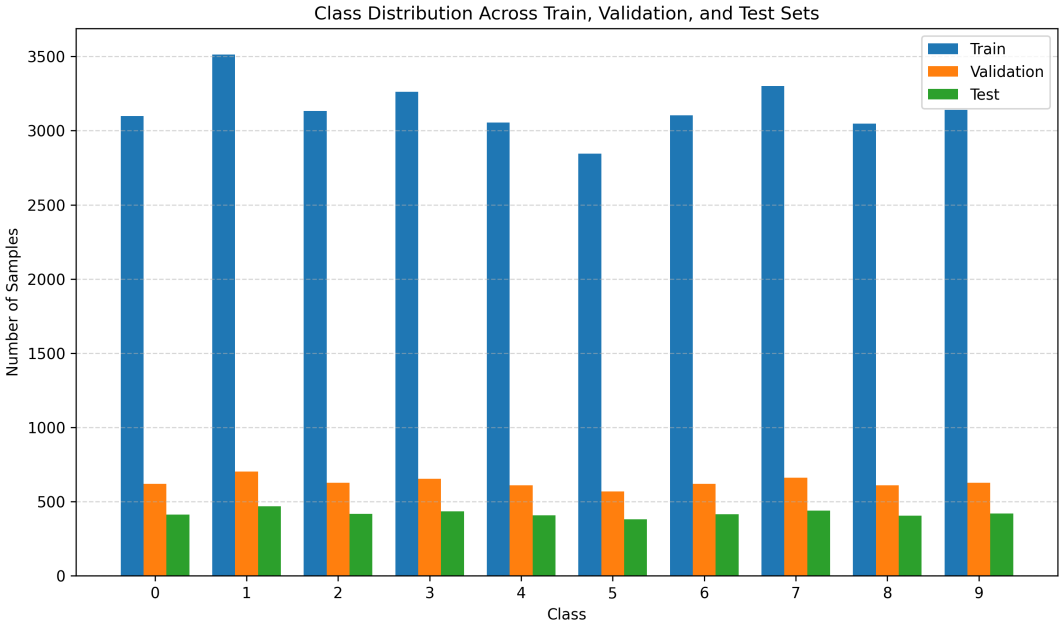


Figure 25: Class imbalance of the dataset

4.3.2 MNIST classification with case B

In `case == b`, the model is a pretrained convolutional neural network (AlexNet). Unlike the MLP in `case == a`, which flattens images and treats pixels independently, AlexNet preserves spatial structure and uses convolutional filters to capture local patterns such as edges and textures. Additionally, AlexNet benefits from transfer learning by using pretrained ImageNet features and fine-tuning only the final layer, providing a strong prior and better generalization with limited data. In contrast, the MLP is trained from scratch, making it more prone to underfitting or overfitting. As a result, case-b is expected to perform better.

Performance improved for class 0, as shown in Table 3. However, issues remain due to color mismatch, where using a convolutional neural network, the model still fails to generalize when red zeros are absent from the training set.

Applying the same improvements as in `case == a`, the final model results are shown in Table 4. All classes achieved over 97% accuracy, recall, specificity, and precision, indicating that class imbalance was

Table 3: Classification results for case B, rounded to 3 decimal places

Class	Accuracy	Sensitivity (Recall)	Specificity	Precision
0	0.903	0.018	0.999	0.776
1	0.998	0.994	0.999	0.992
2	0.938	0.993	0.932	0.619
3	0.999	0.990	0.999	0.995
4	0.994	0.999	0.994	0.945
5	0.968	0.996	0.965	0.741
6	0.999	0.994	1.000	0.999
7	0.999	0.996	0.999	0.995
8	0.998	0.990	0.999	0.994
9	0.998	0.991	0.999	0.990

effectively addressed. Compared to the MLP, the CNN is less affected by imbalance due to its spatial inductive bias and use of pretrained features, which allow it to generalize better even when some classes have fewer examples.

Table 4: Best classification results

Class	Accuracy	Sensitivity (Recall)	Specificity	Precision
0	0.999	0.998	1.000	0.998
1	0.999	0.994	1.000	1.000
2	0.998	0.998	0.998	0.981
3	0.998	0.982	1.000	0.995
4	0.999	0.998	1.000	0.998
5	0.999	0.997	0.999	0.995
6	0.999	0.993	1.000	0.998
7	0.997	0.984	0.999	0.989
8	0.996	0.985	0.997	0.973
9	0.996	0.981	0.998	0.981

5 Conclusion

This report explored classical image analysis, image denoising and inpainting, and the evaluation of image quality using various metrics. While traditional methods can effectively classify and restore images, emerging machine learning techniques offer greater potential for automation and generalization. A key takeaway is that the choice of evaluation metric is critical for accurately assessing reconstruction quality.

6 Appendix

Generation tools such as ChatGPT and Claude.ai are used for writing and debugging code. It was also used to facilitate LaTeX formatting for figures and tables, and to perform spellchecking.

References

[1] Chirayu D. Athalye, Kunal N. Chaudhury, and Bhartendu Kumar. On the contractivity of plug-and-play operators, 2023. Revised version (v3), Dec 17, 2023.

[2] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 International Conference on Pattern Recognition*, pages 2366–2369, 08 2010.

[3] Md Shariful Islam. Plug-and-play prior, admm, regularizer as denoiser (red), and mace. ://mdw771.github.io/2020/07/10/admm-pnp-red.html: :text=Although Accessed: 2025-06-28.

[4] MathWorks. *PIQE: Perception-based Image Quality Evaluator*. The MathWorks, Inc., 2023. Accessed: 2025-06-28.

[5] Pravin Nair and Kunal N. Chaudhury. Averaged deep denoisers for image regularization. *Journal of Mathematical Imaging and Vision*, 2024. Preprint available at arXiv:2207.07321.

[6] OpenCV. Grabcut algorithm for foreground extraction. `https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html`, 2024. Accessed: 2025-06-28.