CS 7180 Project Report Assignment 2

Team Members:

1. Aditya Varshney
2. Luv Verma

GitHub Link: https://github.com/adityav1810/cs7180-assignment2

Abstract

This work experiments on the idea introduced by Hu, Wang, and Lin paper [2] which introduced CNNs tailored for color constancy (termed as FC4) for precise illumination color determination. FC4 attributes different weights to image patches based on their relevance for color constancy estimation and these are incorporated in a unique pooling layer to consolidate local estimations into a global output. Since, the paper talks about confidence and takes them out from the feature maps, we **introduced self-attention layers** to experiment about whether they are better at providing confidence, however our initial results do not suggest so. Our analysis is based on Gehler's Raw Dataset, where the total dataset size is 568. However, due to training constraints, we used a

dataset of 104 images with 70% as a training set and 30% as a test set. The results show that the model without attention performs better in terms of training and validation losses.

Introduction

1. **Problem with Existing Solutions**: The conventional patch-based Convolutional Neural Networks (CNNs) used for color constancy suffer from "estimation ambiguity." This means that small patches or segments of an image might not provide enough information to determine the accurate color of illumination. This ambiguity can lead to inaccuracies in the training and inference phases of the neural network.
2. **Proposed Solution**: The article presents a fully convolutional network architecture designed to address this issue. This network assigns different confidence weights to patches throughout an image based on their relevance for estimating color constancy. The system then merges local estimates into a global solution using a unique pooling layer.
3. **Benefits of the New Approach**:
    - It can determine the importance of different parts of an image for color constancy without additional manual input.
    - The new network design allows for end-to-end training.
    - It processes images of varying sizes efficiently.
    - Demonstrates better accuracy and efficiency on standard benchmarks compared to previous methods.
4. **Understanding the Problem**: The article provides an illustrative example showing how different patches within an image can be ambiguous or informative. For instance, an image patch that contains objects like a banana, which has a recognized color, provides more reliable cues for color constancy than a generic wall which can be of any color.
5. **Comparing with Other Techniques**:
    - The article mentions other CNN-based methods for color constancy. Some use local patches as input while others use the entire image. However, many of

these approaches have limitations like the absence of semantic context, or the need to resize images to fit predefined dimensions.

- The new method, named FC4, considers all image patches simultaneously, which allows it to evaluate the importance of different patches during training.

6. **Significance of Fully Convolutional Structures**: Fully Convolutional Networks (FCNs) are beneficial for tasks requiring pixel-wise output. The proposed network in this article differs from conventional FCNs. Instead of upsampling to produce pixel-wise output, it uses a novel pooling layer to merge the feature map into a single output.

Methods and Architecture

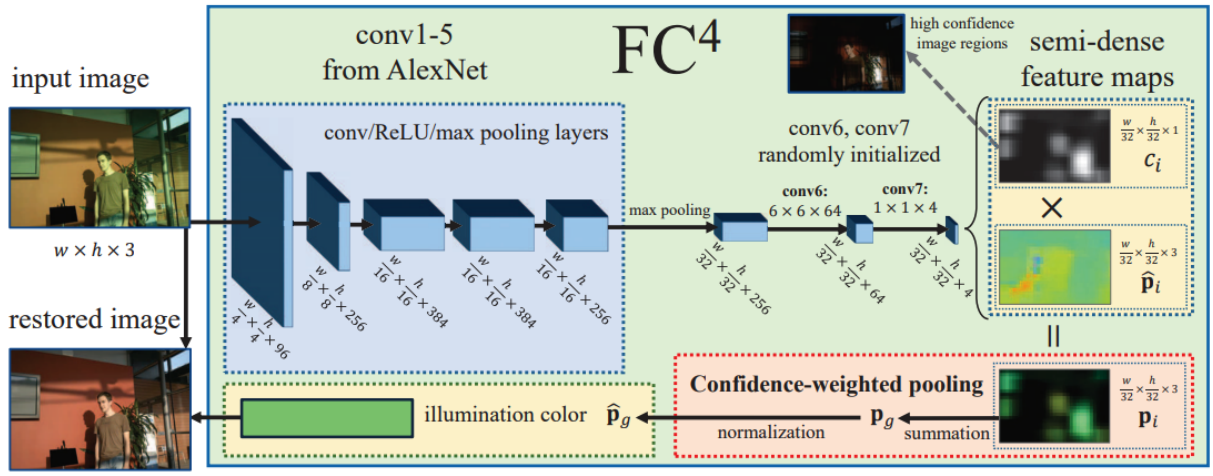


Figure 2: The architecture of AlexNet-FC^4. Replacing AlexNet (conv1-conv5) with SqueezeNet v1.1 (conv1-fire8 plus an extra 2×2 pooling) yields SqueezeNet-FC^4.

Figure 1: AlexNet-FC4 Architecture

1. **Input Image:** The system accepts an input image with dimensions w x h x 3. Here, 'w' denotes width, 'h' denotes height, and '3' represents the three RGB channels. This color information forms the basis for further processing and is crucial for accurate feature extraction and restoration.
2. **Conv1-5 from AlexNet:**

- These layers act as the foundational feature extractors, inspired by the renowned AlexNet architecture.
- The convolutional layers handle the raw image, filtering it to highlight different features and patterns. The incorporation of ReLU activation ensures non-linearity, helping the network learn more complex patterns.
- The max-pooling layers progressively reduce the spatial dimensions of the feature maps, while preserving the most important information, allowing for efficient computation and reduced risk of overfitting.

3. **FC4 (Fully Connected 4):**
    - **Conv6 and Conv7:** These are not mere extensions of AlexNet but are additional layers with their weights randomly initialized. Conv6, with its 6 x 6 x 64 dimension, focuses on capturing granular features. In contrast, Conv7, being 1 x 1 x 4, might be more oriented towards global feature aggregation.
    - **High Confidence Image Regions:** This mechanism ensures that the network concentrates more on areas of the image with clear and reliable features. By doing so, the network can potentially improve accuracy by leveraging these high-confidence regions.
    - **Semi-dense Feature Maps:** Denoted as C_i, these maps maintain a balance between detailed and abstract representation. The spatial resolution of w/32 x h/32 aids in preserving spatial integrity while ensuring computational efficiency.
4. **Confidence-weighted Pooling:**
    - Unlike traditional pooling methods that often lose certain feature details, this innovative technique incorporates a weighting mechanism. By associating weights based on feature confidence, it ensures that significant features are preserved and emphasized.
    - The final pooled feature map, P_i, with dimensions w/32 x h/32 x 3 channels, is a balanced representation, optimized for subsequent processing.
5. **Illumination Color (P_9):**
    - Post-processing, the network provides an output termed as "Illumination Color". This is not just a by-product but a vital output, indicating the perceived lighting condition of the input image. Understanding illumination can be essential for

applications where lighting plays a crucial role, like photography or film production.

6. **Restored Image:**
    - The culmination of the process is the restored image. This output is a testament to the network's ability to enhance, restore, or modify the input image based on the learned features and the applied transformations.

Adaptation to SqueezeNet:

The architecture can be further adapted by replacing AlexNet layers (Conv1-Conv5) with SqueezeNet v1.1 layers (Conv1-Fire8 plus an extra 2 x 2 pooling). This version is termed "SqueezeNet-FC4."

Hyper Parameters:

Learning Rate: 0.0003

Batch Size: 1

Epochs: 2000

Major Modification: Introduction of self-attention Layers in Squeeze-Net

1. **Self-Attention Modules**:
    - Four self-attention modules are initialized in the SqueezeNet class: `attention1`, `attention2`, `attention3`, and `attention_classifier`.
    - Each self-attention module receives a specific number of input channels, which corresponds to the number of channels in the feature maps to be processed
2. **Position in the Architecture (version 1.0)**:
    - `attention1` is placed after the first convolutional layer and before the first `Fire` module.
    - `attention2` is placed after the third max-pooling layer and before the fourth `Fire` module.
    - `attention3` is placed after the fifth max-pooling layer and before the seventh `Fire` module.
    - `attention_classifier` is used just before the final convolutional layer in the classifier.

3. **Position in the Architecture (version 1.1)**:
    - Note: In the 1.1 version of SqueezeNet provided, the self-attention modules are not used. The architecture consists of convolutional, pooling, and `Fire` modules only.
4. **Usage in the Classifier**:
    - `attention_classifier` is incorporated into the classifier sequence. It processes the feature maps before they pass through the dropout layer, the final convolutional layer, and the subsequent operations that generate the class scores.
5. **Number of Uses**:
    - In version 1.0 of the SqueezeNet architecture, the self-attention mechanism is applied four times: thrice in the feature extraction sequence and once in the classifier sequence.
    - In version 1.1, self-attention is not used within the feature extraction, but it is still applied once in the classifier sequence.
6. **Functionality**:
    - The self-attention modules allow the network to focus on different parts of the input feature maps by computing a weighted combination of input features. The weights are dynamically determined based on the similarity between feature pairs. This enables the model to capture long-range dependencies and relations between different parts of the input, enhancing its ability to recognize patterns and details that might be spread out across the image.

Dataset Description

We use the Gehler dataset [1] which contains 568 images of a variety of indoor and outdoor shots using high-quality cameras. These images were initially saved in the RAW format and were non-linear, demosaiced, and also included the effects of the camera's white balancing.

The cameras generate 12-bit data per channel, yielding a digital count range of 0 to 4095. Within the raw images, there are 4082 x 2718 values for the Canon 1D and 4386 x 2920 values for the Canon 5D, all in a 12-bit RGB pattern.

For color image creation, an average of the two G values was taken, with no subsequent demosaicing. This process results in a linear image of 2041 x 1359 (for Canon 1D) or 2193 x 1460 (for Canon 5D) in camera RGB space, with a gamma of 1. Furthermore, each image has a MacBeth ColorChecker for reference.

Dataset Preparation

We set the dataset for the experiment as follows:

1. **Data Collection**:
    - Download 12-bit PNG images from [1] into the `dataset/images` folder.
    - Extract corresponding true illumination data from `dataset/metadata.txt`.
2. **Masking and Saving**:
    - Using the preprocessing tools, mask the MacBeth ColorChecker in each image with a black polygon to ensure the neural network won't use it as visual cues during training or testing.
    - The images without the color checker are saved in numpy format within the `dataset/preprocessed/numpy_data` directory.
    - Corresponding illumination data (R, G, B values) are stored as numpy labels in the `dataset/preprocessed/numpy_labels` directory.
3. **Image Corrections & Format Conversions**:
    - Convert these images to a visible linear format, ensuring consistent representation across all images.
    - The linear images are saved in the `dataset/preprocessed/linear_images` directory.
    - Subsequently, correct these images for illumination biases and discrepancies.
    - Store the illumination-corrected images in the `dataset/preprocessed/gt_corrected` directory, serving as ground truth references for model evaluation.
4. **Augmentation & Transformation**:
    - To enrich the dataset and make the neural network robust to variations, the images undergo a series of augmentations. This includes operations like rotation, random cropping, color intensity adjustments, and scaling.

- Depending on whether the images are for training or testing, they are either augmented using the `Augment` class or simply resized to match the neural network's expected input dimensions.

5. **Data & Label Storage**:
    - The images, once processed and augmented, are stored in a format suitable for neural network consumption. They are saved as numpy arrays in specified directories to ensure easy access during the training and testing phases.

Loss Function

The dot product between the normalized predicted tensor and the normalized label tensor is computed. This dot product gives a measure of the cosine of the angle between the two tensors. It represents the cosine similarity between the predicted and label tensors. The dot product is clamped within the valid domain of [-1,1] and is fed to the inverse cosine (`acos`) function. The upper bound of the dot product is set very close to 1 (i.e., `0.999999`) to avoid potential computational issues. The inverse cosine (`acos`) of the clamped dot product gives the angular difference between the predicted tensor and the label tensor in radians. The average of all the computed angular differences is taken. This gives a single scalar value representing the mean angular loss between the predictions and the labels.

Results

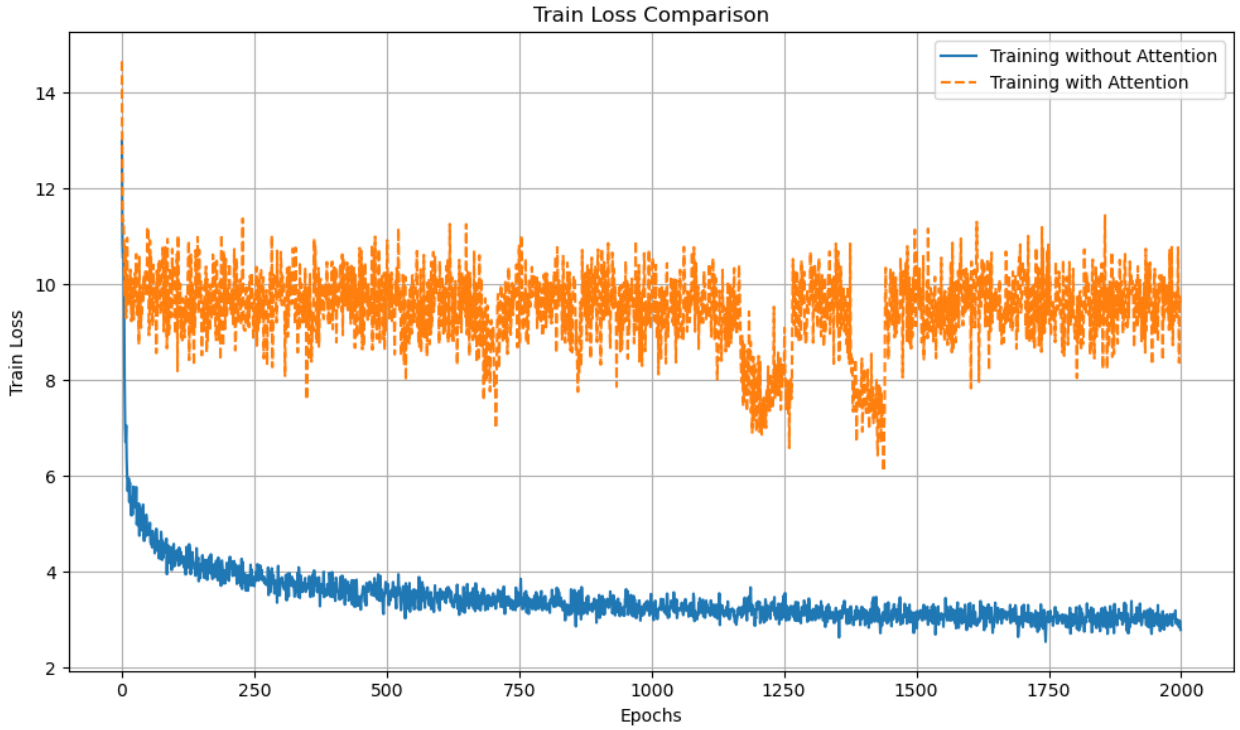
Metrics	Without Attention	With Attention
best_mean	1.733	3.313
best_median	1.266	1.142
best_trimean	1.348	1.354
best_bst25	0.445	0.343
best_wst25	3.53	8.595
best_wst5	4.196	14.904

- **best_mean**: This metric measures the average performance of the best results. For the models, the one **without attention** has a performance of 1.733, while the one **with attention** performs at 3.313. This indicates that the average of the best performances is better for the model without attention.

- **best_median**: Represents the middle value of the model's best results. The model **with attention** has a value of 1.487, while the model **without attention** has a best median of 1.266. Given the limited training on the attention layers, it seems that the model is able to generalize to some extent.
- **best_trimean**: This metric averages the best results' median with their mid-range. The model **with attention** registers a value of 1.354. while the model without attention has a value of 1.348. This indicates that both models provide competitive results.
- **best_bst25**: Reflects the best 25% of the model's top results. Interestingly, the model **with attention** stands at 0.343 which is an improvement from the model with no attention layers. This indicates that attention works well in the top quartile of data.
- **best_wst25**: Contrary to best_bst25, this indicates the worst 25% of the model's top results. The model **with attention** is at 8.595, however, the metric for the model without attention layer seems to be more robust. This indicates that the model without attention layers is able to generalize well in all edge-case scenarios.
- **best_wst5**: This denotes the worst 5% of the model's top results. For the model **with attention**, this value is notably high at 14.904, suggesting that in its worst cases among the best results, it struggles significantly.

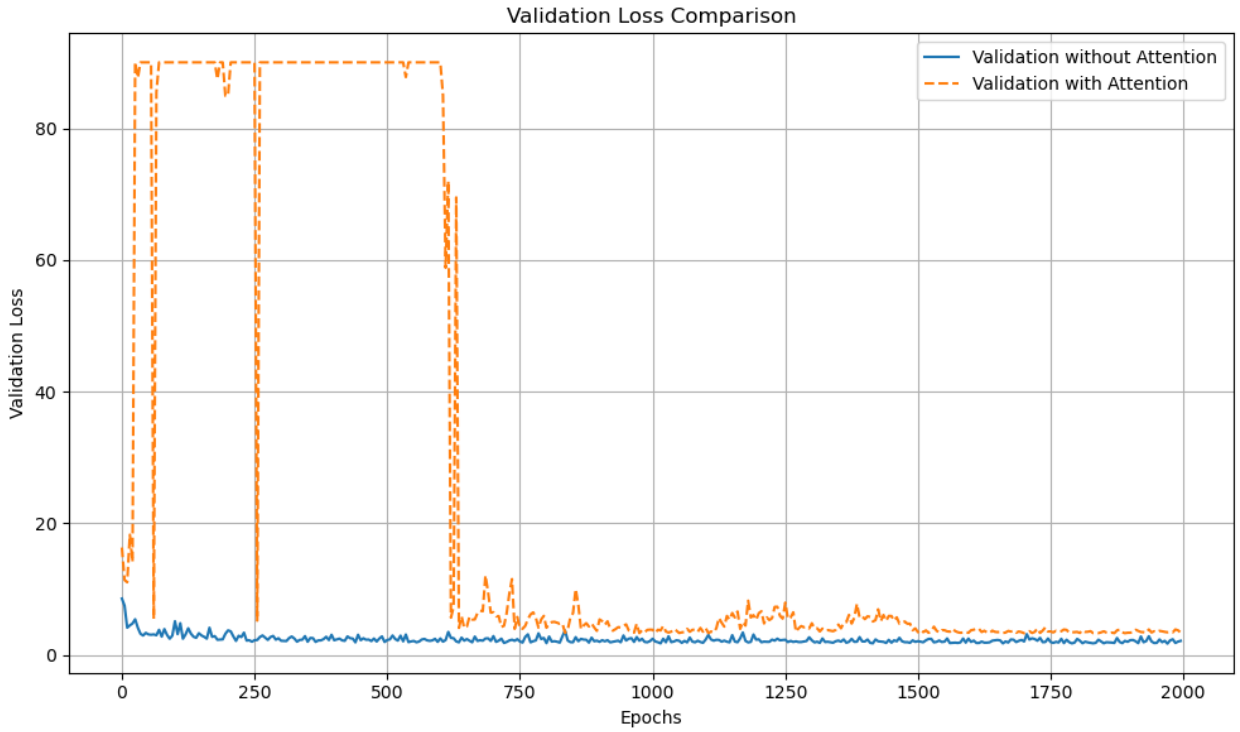
In summary, while the attention mechanism often improves performance, its benefits here seem mixed. The model **with attention** does perform slightly better in certain metrics like best 25% performances. However, it grapples notably in its worst-case scenarios, which may imply that the attention mechanism might be over-focusing on certain aspects, leading to errors in specific situations.

Train Loss Comparison



- **Graph Overview**: The first graph depicts the train loss comparison between two different models: one trained without attention and the other with attention. The x-axis denotes the number of epochs (from 0 to 2000) while the y-axis represents the training loss.
- **Without Attention**: The solid blue line that shows a consistent and smooth decrease in training loss, eventually stabilizing around a certain value. This indicates that the model has learned effectively over time and has reached a point of minimum error on the training set.
- **With Attention**: The dashed orange line, on the other hand, seems to have a more erratic pattern with fluctuations throughout the training process. This could suggest that the attention mechanism adds a level of complexity or instability to the training, or it may be sensitive to certain data patterns.

Validation Loss Comparison



- **Graph Overview**: The second graph showcases the validation loss comparison. Similarly, epochs are on the x-axis and the validation loss values are on the y-axis.
- **Without Attention**: The solid blue line starts with a higher loss but quickly stabilizes after a few epochs. It remains quite stable throughout the training process with minor fluctuations, suggesting that the model generalizes well on unseen data.
- **With Attention**: The dashed orange line starts with a very high loss which drops rapidly. However, post this, there are significant spikes, indicating moments of high error. The pattern here is more unstable compared to the training loss for the attention model. The frequent peaks might indicate that the model with attention overfits the training data and struggles to generalize on the validation set.

In conclusion, in this particular case, the use of attention seems to introduce more variability in the loss The performance metrics also indicate that attention does not consistently outperform the non-attention model. This suggests that careful consideration and potentially more fine-tuning are needed when implementing attention mechanisms for specific tasks.

Image ID: 8D5U5524 | Error: 0.5808

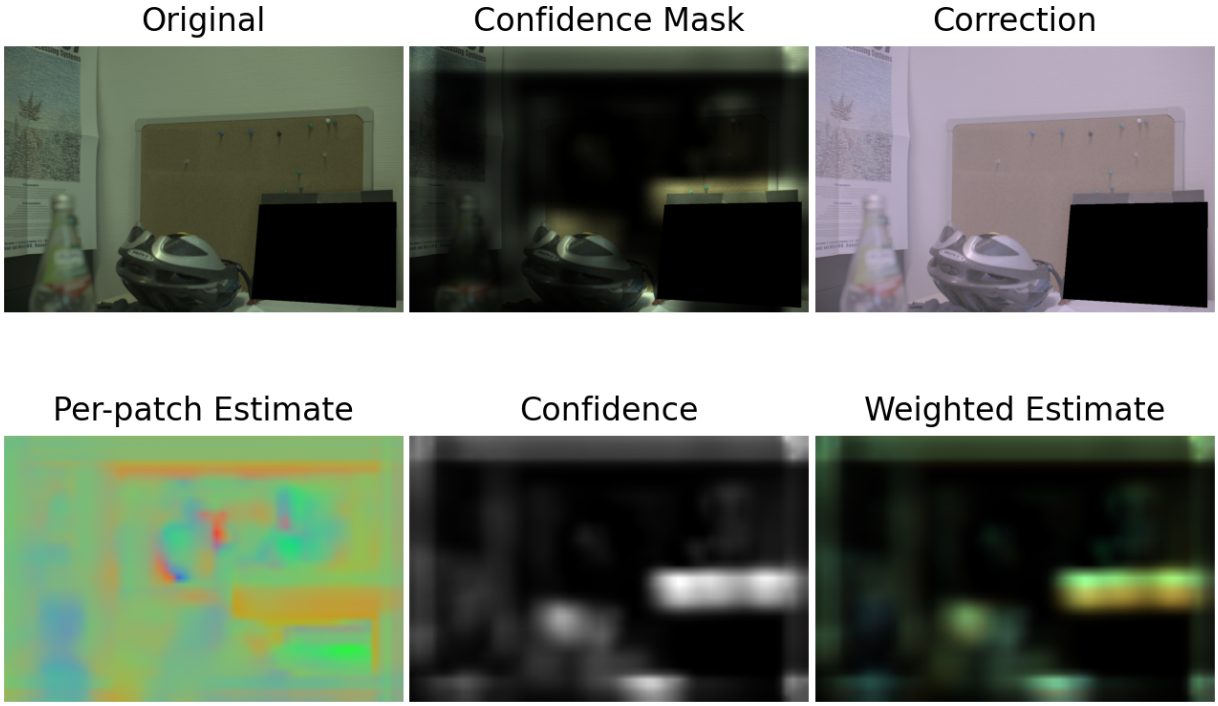


Image ID: 8D5U5526 | Error: 0.1998

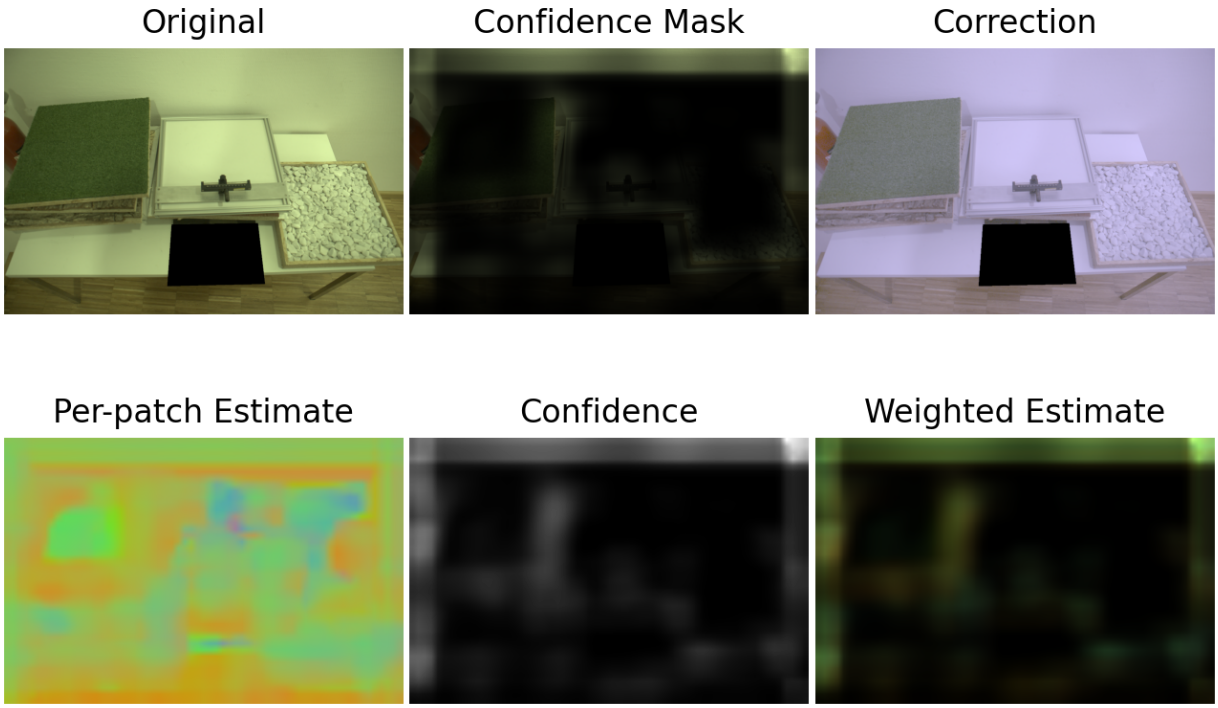


Image ID: IMG_0281 | Error: 1.7162

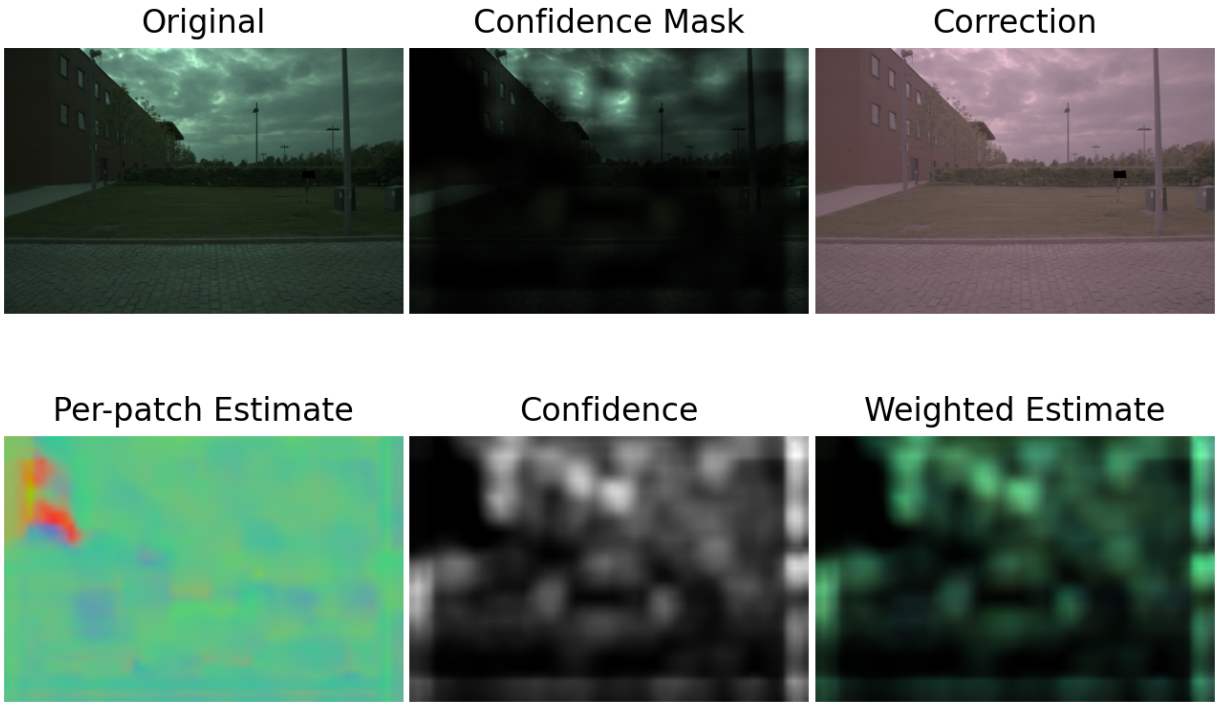


Image ID: IMG_0285 | Error: 2.0142

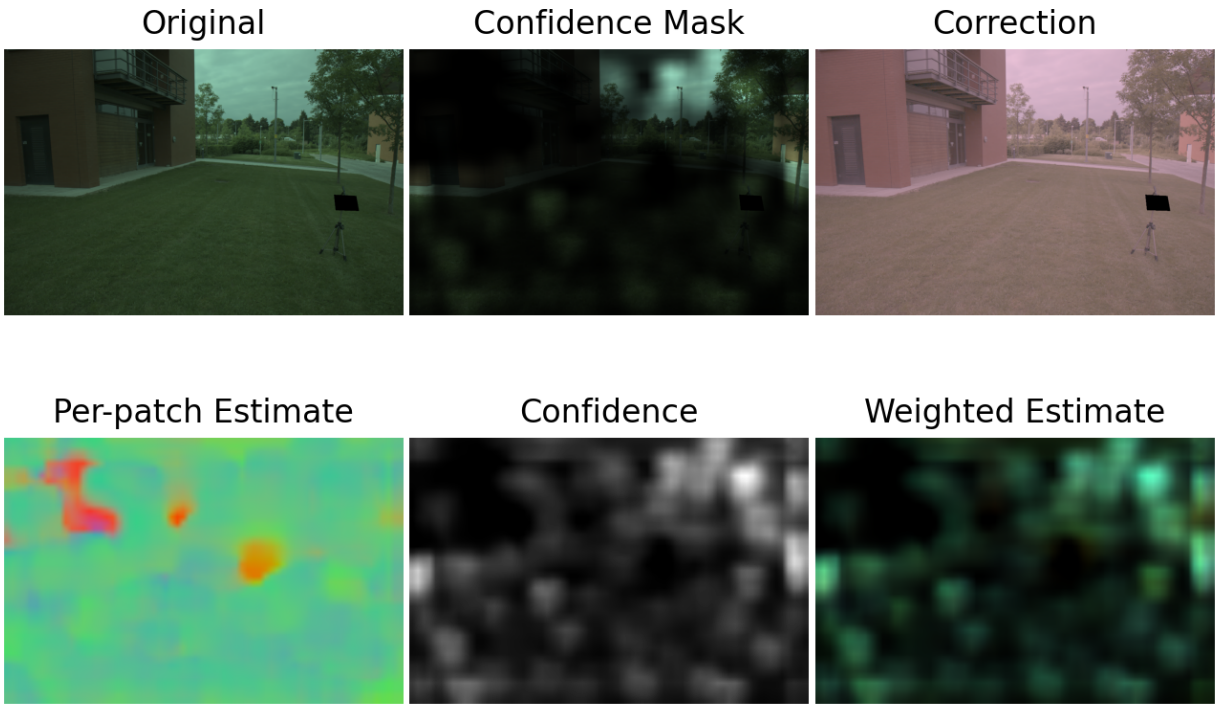


Image ID: IMG_0389 | Error: 1.5657

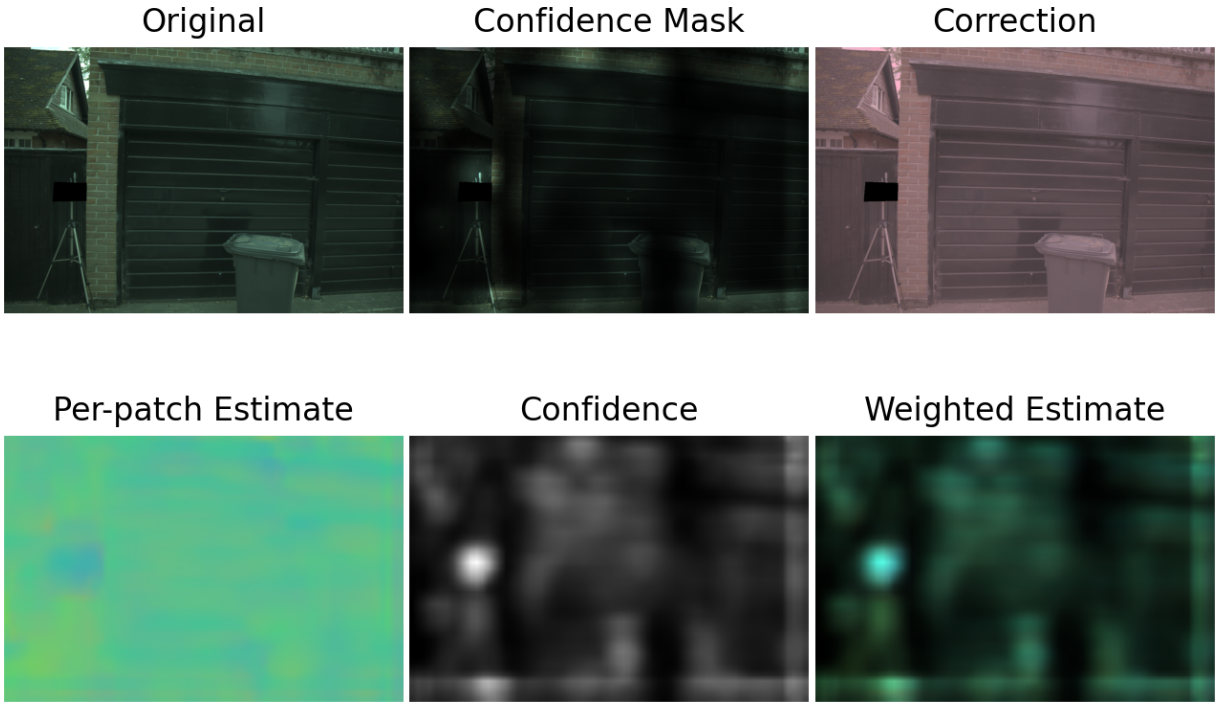


Image ID: IMG_0382 | Error: 1.4931

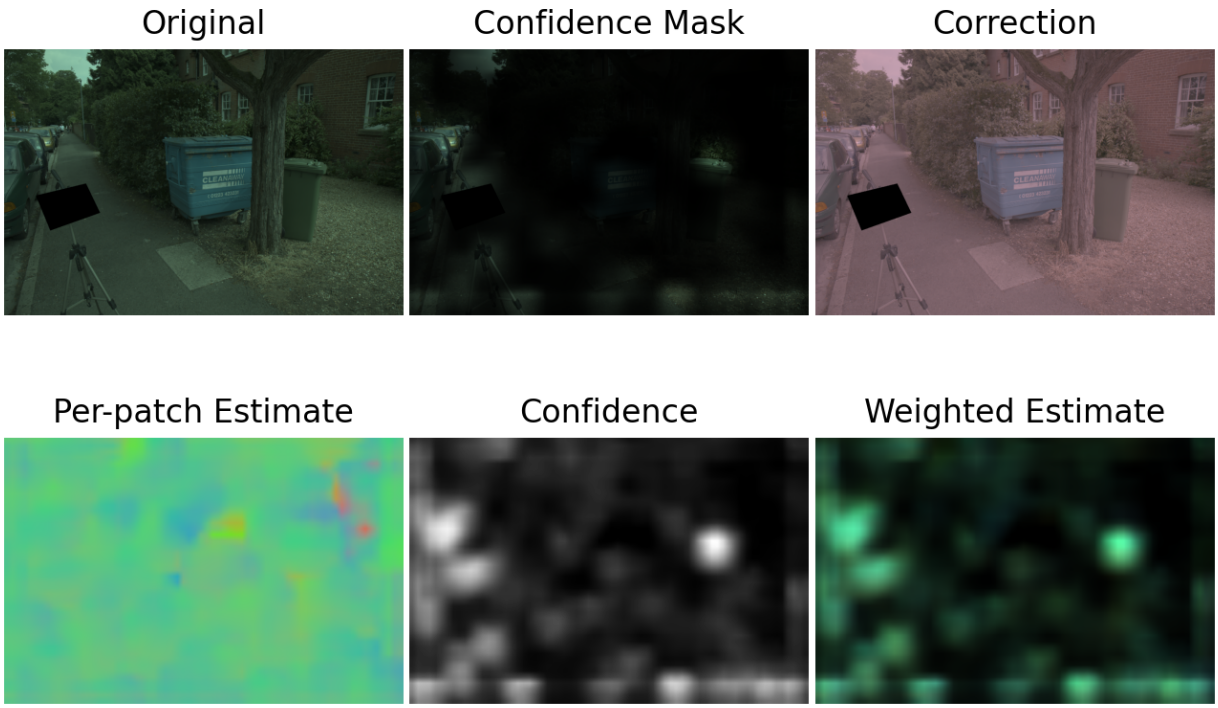


Image ID: IMG_0380 | Error: 1.7089

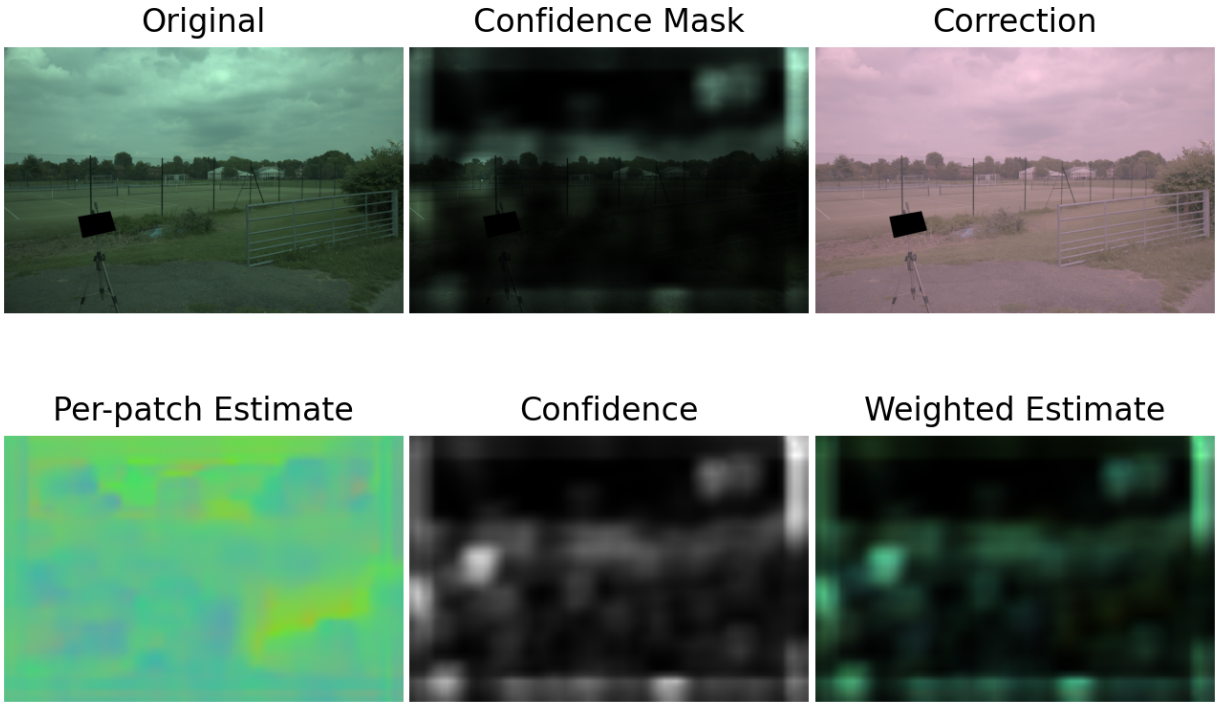


Image ID: IMG_0365 | Error: 0.8116

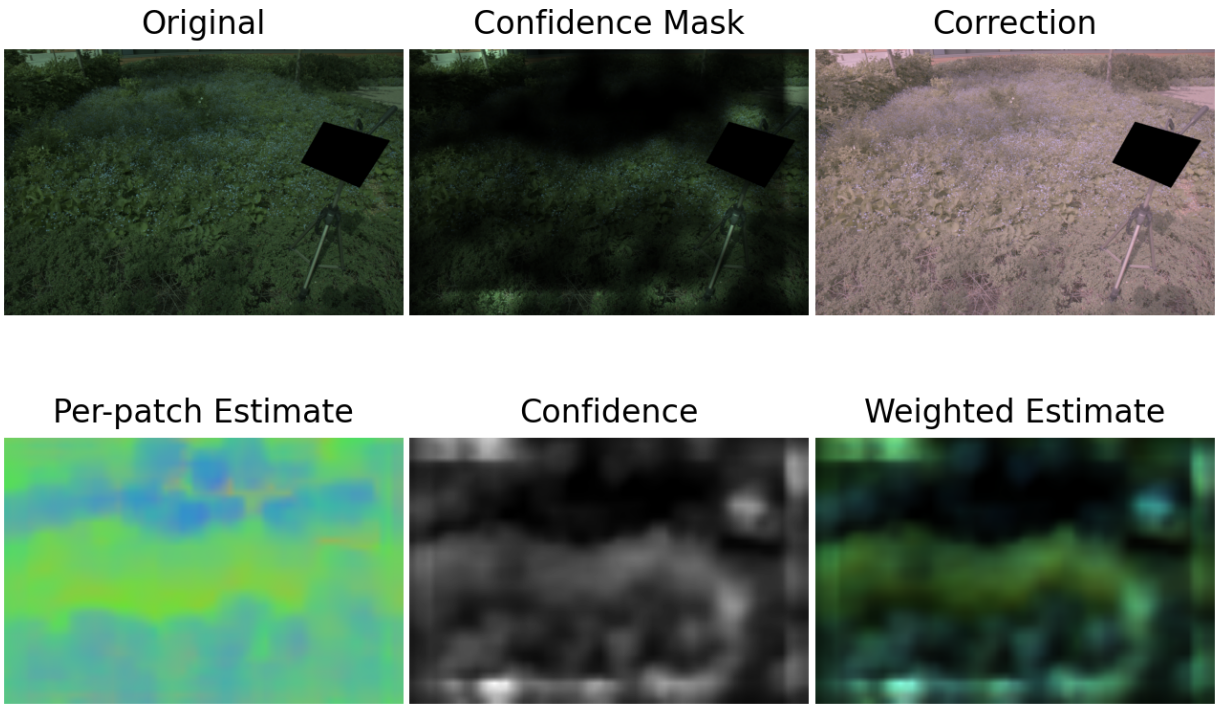
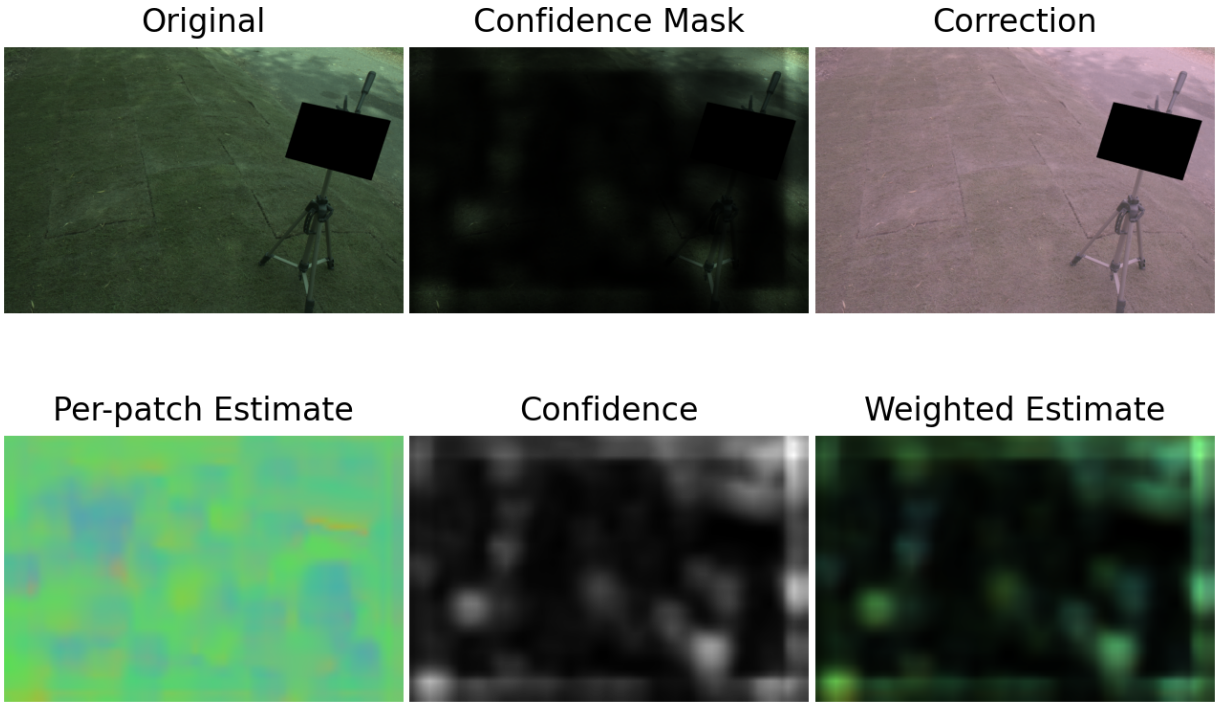


Image ID: IMG_0345 | Error: 5.0111



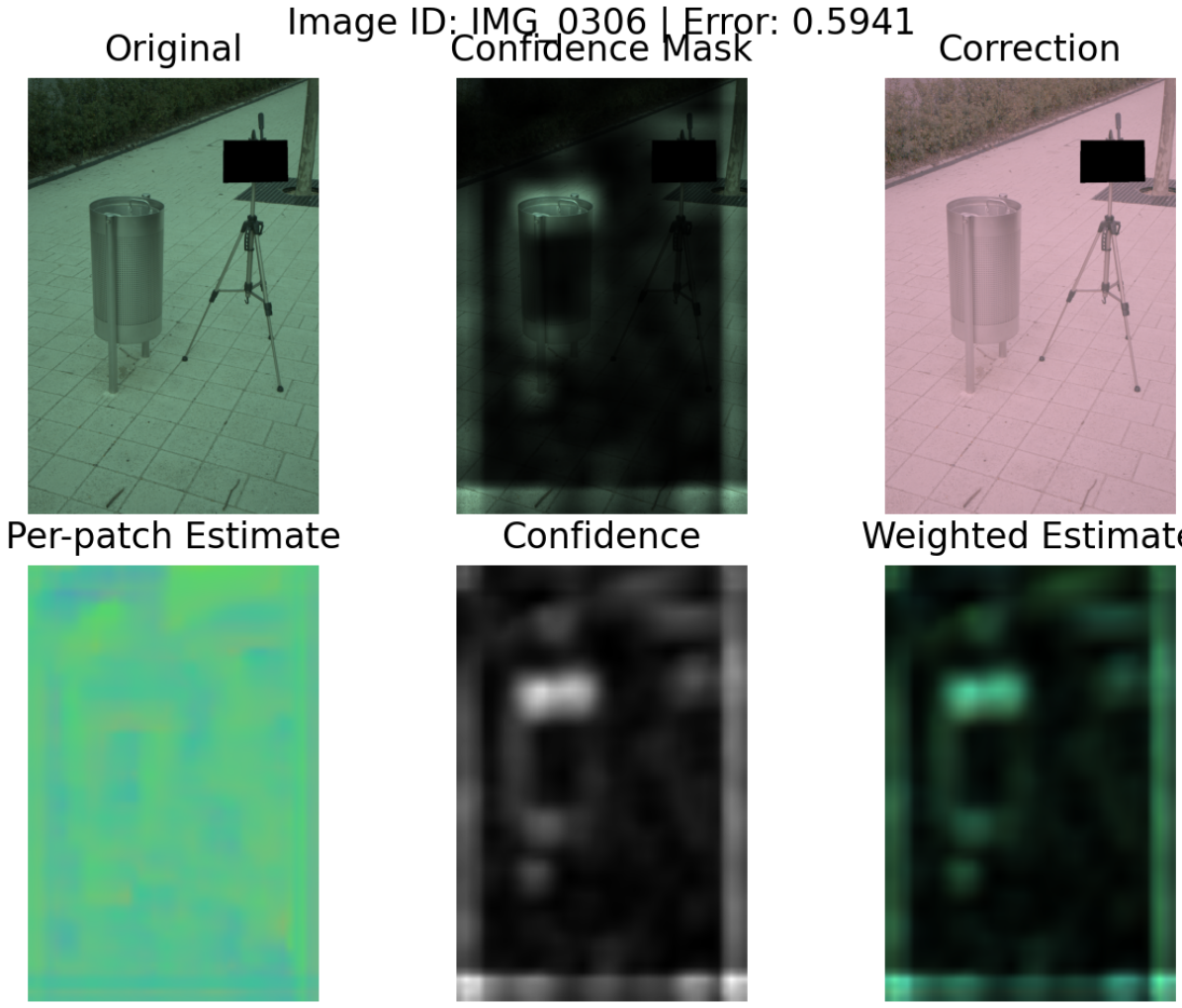


Image ID: IMG_0306 | Error: 0.5941
Original
Confidence Mask
Correction
Per-patch Estimate
Confidence
Weighted Estimate

Image ID: IMG_0292 | Error: 1.0774

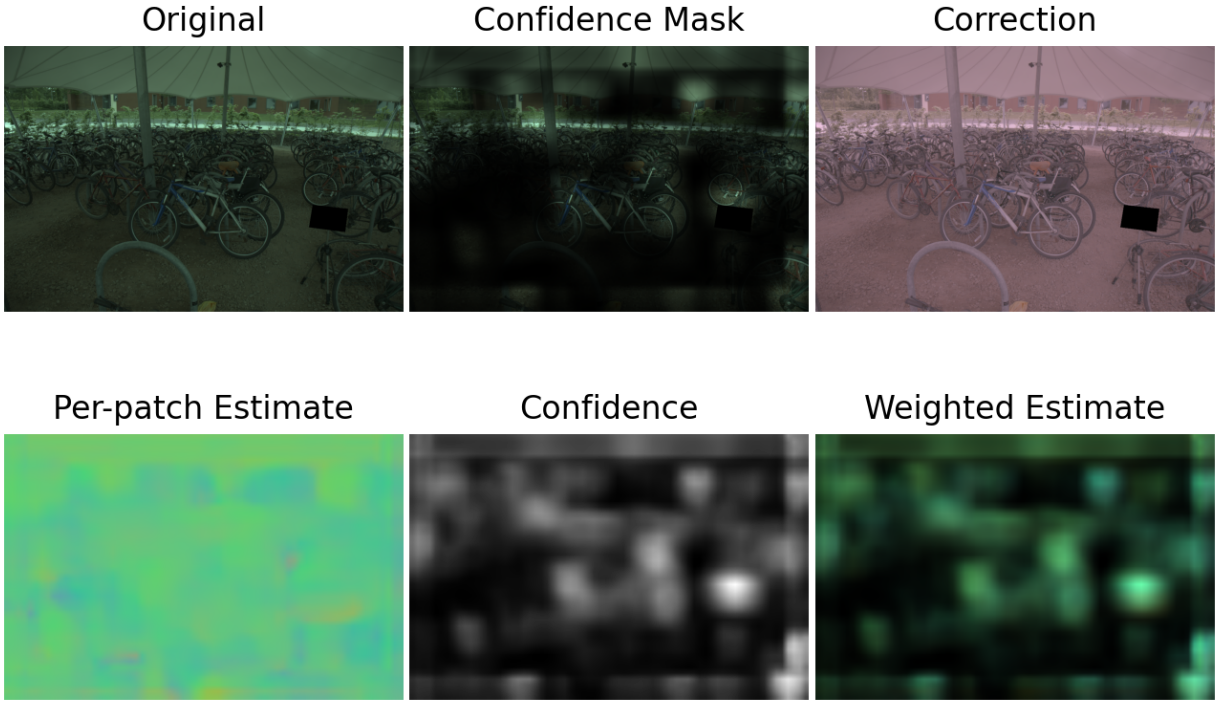


Image ID: IMG_0369 | Error: 1.1919

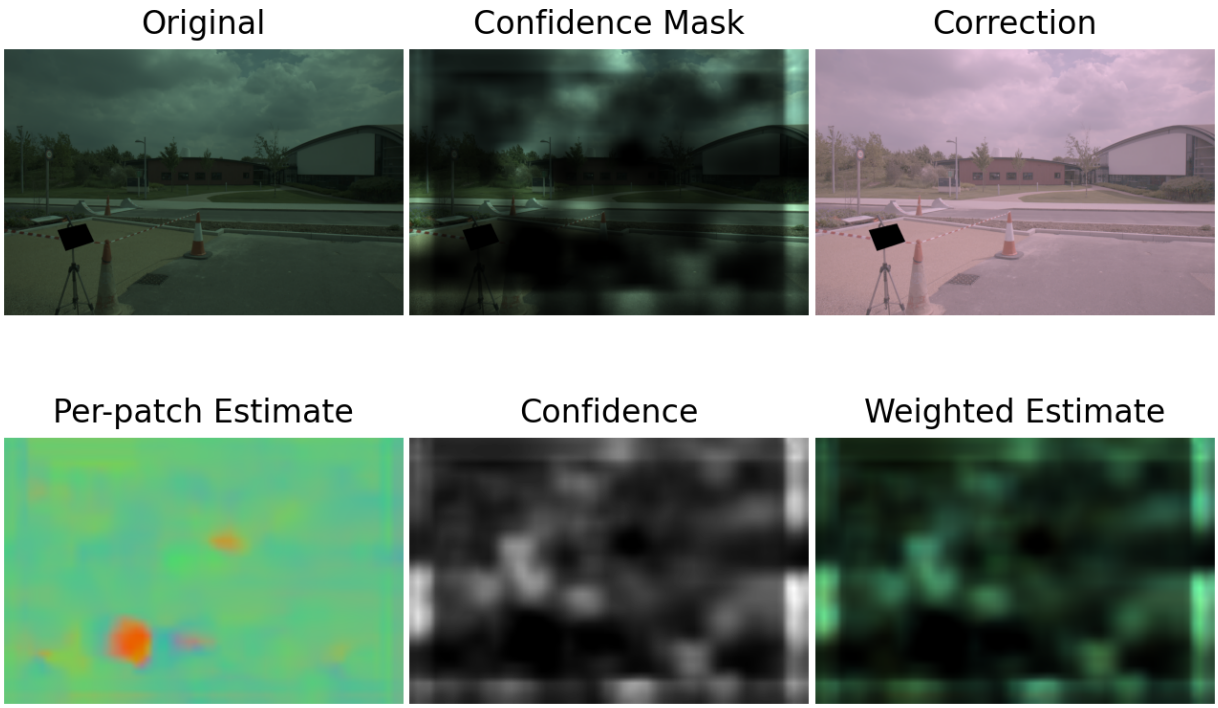


Image ID: IMG_0361 | Error: 1.8609

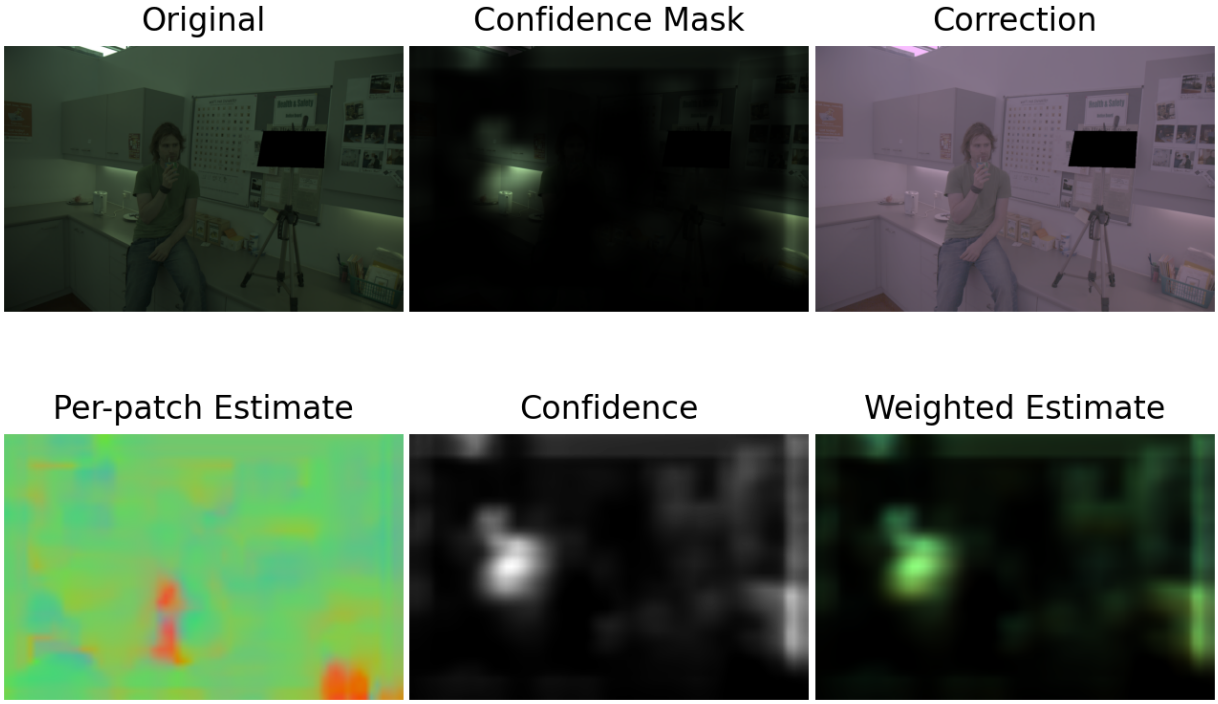
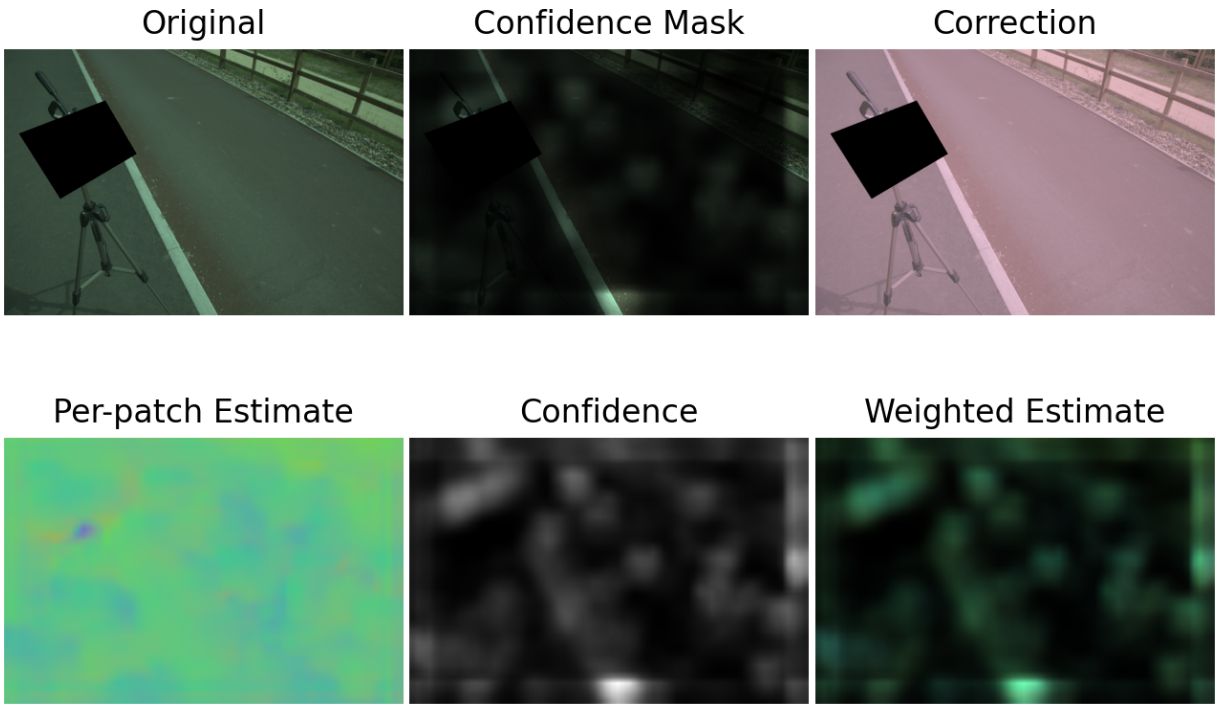


Image ID: IMG_0354 | Error: 1.5818



Observations and Discussions

1. **Per-patch Estimate**:
    - This is a raw data interpretation step. Instead of processing an entire image at once, the system breaks the image down into smaller, manageable "patches." This allows for a more detailed and localized analysis. For each patch, the system gives its best guess on what it represents, without yet taking into account the surrounding context or any reliability metrics.
2. **Confidence Visualisation**:
    - After creating the initial estimates, the system then evaluates how much it trusts these estimates. This self-assessment is crucial because not all parts of an image are equally clear or decipherable. Some areas might have ambiguous data, noise, or other factors that make interpretation difficult. The confidence visualization offers a way to visually represent the system's level of trust in its initial guesses.
3. **Weighted Estimate**:
    - This is where the initial guess and the system's trust in that guess come together. Using the confidence visualization as a guide, the system refines its per-patch estimates. If the system is highly confident about an estimate, that estimate remains largely unchanged. However, if there's low confidence, the estimate might be adjusted, taking cues perhaps from neighboring patches or broader patterns in the image.
4. **Confidence Mask**:
    - The confidence mask is a synthesized view of the overall reliability of the image's data. While the confidence visualization focuses on individual patches, the mask looks at the bigger picture. It provides an at-a-glance view of which regions in an image are most trustworthy and which ones are more ambiguous.
5. **Interrelation between Per-patch Estimate & Confidence**:
    - The nature of the initial estimates can directly influence the confidence levels. Areas with more variation, complexity, or noise might be more challenging for the system to interpret reliably. As a result, these areas often correlate with

lower confidence. Conversely, patches with clear, distinct features or patterns are easier to estimate, leading to higher confidence levels.

6. **Confidence Mask & Weighted Estimate**:
    - Visualizing it, the brighter regions in the confidence mask (indicating high confidence) would be where the system has a higher degree of trust in the information. When this high-confidence data gets translated into the weighted estimate, it holds greater weight or influence. Imagine a painting where certain areas are illuminated by a spotlight (high confidence) – these areas would naturally attract more attention and would likely remain more vivid in the final representation. Conversely, darker regions (lower confidence) in the mask may result in more subdued or altered representations in the weighted estimate, as the system would be more cautious with these areas.
7. **Confidence & Weighted Estimate**:
    - When a patch in the image has high confidence, the system trusts its initial interpretation, meaning the weighted estimate remains quite similar to the per-patch estimate. On the other hand, for patches with low confidence, the system might look to adjacent patches or broader patterns for guidance. These areas in the weighted estimate might seem more blended or averaged out as the system takes a more conservative approach, not trusting its initial guess fully.
8. **Darker Areas in Per-patch Estimate**:
    - These darker areas might suggest regions where there's ambiguity in data interpretation. Visual factors like shadows, glares, or intricate textures can make it challenging for algorithms to decipher and interpret information accurately. Thus, the system might lower its confidence in these areas, leading to more caution in the weighted estimate.
9. **Brighter Areas**:
    - Just as some patches might be ambiguous, others can be crystal clear. Brighter areas in the per-patch estimate could represent zones where the system is more certain about the data it's analyzing. Distinctive features, clear patterns, or recognizable textures would boost the system's confidence in these regions. The system would likely have a higher trust level for these patches, influencing the final weighted estimate substantially.

Challenges

Hardware Constraints

As we have limited access to GPU compute resources, we were limited to fitting out the entire model and training data in a single 32GB GPU. Furthermore, attention layers require intensive computing resources and hence we were limited to using a subset of the actual training data. Out of the 568 images available in the original dataset, we randomly chose 104 images for training and testing purposes. As the image size itself is relatively large (4082 x 2718), we had to limit the batch size to 1 in order to train the network for 2000 epochs. Furthermore, we also wanted to get validation at train time in order to understand the model performance which further led to the model training time exceeding the maximum resource allocated time of 8 hours.

Idle Time Overheads

Due to a single GPU access and a limited 8-hour duration for the Cluster, we could not train the model for a desired amount of time and with a batch size of more than 1. Moreover, getting constant access to a discovery cluster with the required amount of compute memory was a challenge.

Training Data Dilemma

The initial challenge we faced for this project was to set up and download the dataset. Self-attention layers are computationally expensive to train and require large computing resources as well as a longer training time to train. With limited access to computing, training with attention layers could only be limited to 2000 epochs. This led to the model overfitting on the training set. This could be observed by a considerable difference between train and validation loss.

Conclusion, Reflection, and Acknowledgments

This project helped us learn about Color Constancy and Intrinsic imaging in great depth. We started by exploring the various datasets available and learned how they were created while also realizing the limitations as well. The use of the Macbeth color checker was really intriguing and it was insightful to see how they play such an important role in creating ground truth labels. We also learned about using an existing trained neural network, dissecting it, adding pre-trained weights manually, and adding new layers in

between the trained layers. This was particularly challenging as it required a lot of experimentation and running inferences on the model multiple times. We also developed a better intuition of attention layers and why it is essential to put them strategically in the model. Furthermore, we learn how confidence-based pooling is an effective way to use local estimates and help the neural network in learning useful data instead of noise. Finally, we also learned how to follow best practices when it comes to saving metrics and model weight.

This project is based on the research study conducted by Hu et al [2]. And the codebase is built on top of the work by Matteo Rizzo [3].

Citations

[1] Shi, L., & Funt, B. (n.d.). Re-processed Version of the Gehler Color Constancy Dataset of 568 Images. Retrieved from http://www.cs.sfu.ca/~colour/data/

[2] Hu, Y., Wang, B., & Lin, S. (2017). Fc4: Fully convolutional color constancy with confidence-weighted pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4085-4094).

[3] https://github.com/matteo-rizzo/fc4-pytorch