Problem 2

Implementation and evaluation of edge detection and Hough transform techniques on the 'house.tif' image, with analysis of the algorithm's performance and investigation of how parameter choices affect the results.

Implementation

Edge Detection

To detect edges in the image, we first convert it to grayscale. Then, we apply a Laplacian of Gaussian (LoG) filter created with sigma and width parameters. This filtering process, implemented using convolution operation, highlights the image's edges.

Hough Transform

After identifying edges with the LoG filter, we use the Hough transform to find lines in the image. This involves calculating an accumulator space that considers different line angles (theta) and distances from the origin (rho). Finally, we employ RANSAC (Random Sample Consensus) to refine the parameters of the detected lines and improve their accuracy, especially in the presence of noise or imperfections.

Results and Analysis

Edge Detection

To evaluate the edge detection algorithm, I varied the sigma and width parameters of the LoG filter. I then visually inspected and analyzed the resulting edge images. As observed in the generated plots, different combinations of sigma and width led to varying edge detection results. Specifically, increasing the width of the Gaussian filter tended to result in smoother edges, while varying the sigma parameter affected the scale of the detected edges.

Hough Transform

I applied the Hough transform to the edge-detected images to identify lines in the 'house.tif' image. The resulting accumulator space visualization served as a heatmap, where brighter regions indicated areas with a higher concentration of lines. This helped me understand the dominant line orientations within the image. I then employed RANSAC to refine the line parameter estimates (such as slope and intercept). This step is crucial, especially in scenarios with noise or imperfections, as RANSAC can help eliminate outliers and improve the overall accuracy of the detected lines.

Conclusion

The implemented edge detection and Hough transform algorithms effectively analyzed the 'house.tif' image. By adjusting the LoG filter's sigma and width parameters, I was able to control the sensitivity and scale of the detected edges. Similarly, varying the parameters within the Hough transform could influence the line detection accuracy.

Problem 3

Denoising an image using the Gaussian Pyramid and Gaussian Filtering methods. This method removes noise from an input image while preserving important details and structures.Here we will denoise the image using a Gaussian pyramid with Gaussian filtering.

Methodology

Gaussian Pyramid and De-emphasizing High-Frequency Components: The Gaussian Pyramid method constructs a hierarchical representation of the image by iteratively applying a Gaussian filter and downsampling. Each level in the pyramid captures a progressively smoother version of the image, effectively separating high-frequency details (often associated with noise) from low-frequency structures (representing the actual image content).

De-emphasizing high-frequency components during reconstruction is crucial for noise reduction. This is achieved by manipulating the information retrieved from the higher levels of the pyramid. Here, we employ two primary strategies:

Gaussian Filter Choice: The Gaussian filter acts as a low-pass filter, attenuating high-frequency components while preserving low-frequency ones. The filter's standard deviation (sigma) controls the level of smoothing. The code (refer to the gaussian_kernel function) defines a 1D Gaussian kernel based on sigma. By choosing a larger sigma value for filtering each level in the pyramid (implemented in the denoise_gaussian_pyramid function), It can effectively reduce the impact of high-frequency noise on the final reconstruction.

Implementation

Gaussian Pyramid: This function constructs a Gaussian pyramid by iteratively applying downsampling to create progressively smaller versions of the image

Gaussian Kernel: This function generates a 1D Gaussian kernel for filtering based on a provided sigma value

Gaussian Filter: This function applies the Gaussian filter defined by the kernel to the input image using a 2D convolution operation

Denoise Gaussian Pyramid: This function performs the core denoising steps:

- Creates a Gaussian pyramid of the noisy image using the gaussian_pyramid function.
- Applies Gaussian filtering on each level of the pyramid with the chosen sigma value.
- Reconstructs the denoised image by upsampling the filtered higher levels and adding them to the lower level, effectively combining information from different scales.

Parameter Selection

The choice of sigma for the Gaussian filter is a trade-off between noise reduction and detail preservation. A higher sigma leads to stronger noise suppression but can also blur edges. The code allows experimenting with different sigma values to find a suitable balance.

Conclusion

Successfully able to denoise the image

Problem 4

The objective of this project is to explore the capabilities of deep learning models for image feature extraction and classification tasks. Specifically, we focus on two popular convolutional neural network (CNN) architectures: VGG16 and AlexNet. We aim to extract image features using pre-trained versions of these models and evaluate their performance in classifying images from the CIFAR-10 dataset.

Implementation

Data Preprocessing: The CIFAR-10 dataset is used for training and testing, containing 60,000 small color images (32x32 pixels each) belonging to 10 different categories. To prepare the images for the deep learning model, several pre-processing steps are applied. These steps include resizing the images, cropping them around the center, converting them to a format the model understands (tensors), and adjusting their color levels to a standard range. This pre-process ensures the model receives data in a format that's easy to work with.

Feature Extraction: To understand the content of the images, pre-trained models like VGG16 and AlexNet come into play. These models act like powerful image recognizers, having already learned a lot from analyzing massive datasets. We can leverage their expertise by feeding our images through their convolutional layers. These layers act like feature detectors, identifying patterns and shapes within the

image. At the end of these layers, the models output a 'feature vector' - a condensed summary of the image's key characteristics learned by the models.

Nearest Neighbor Classification: Extracted image features (using VGG16 & AlexNet) are stored with their labels. A k-Nearest Neighbors (KNN) classifier then takes over. For each new image, it extracts features using the same models and finds the closest matches (based on Euclidean distance) within the stored feature data. This helps classify the new image.

Evaluation

The KNN classifier's performance is measured by accuracy. This compares the predicted labels (KNN's guess) with the true labels of the test images. Separate accuracy scores are calculated for VGG16 and AlexNet to see which performs better.

Results

I tested for below 300 samples as even with GPU it was taking longer time. I researched on google about its evaluation and found that pre-trained models like VGG16 and AlexNet for feature extraction worked really well for classifying images. Their high accuracy suggests they can pick out the important details in an image and apply that knowledge to unseen images. By comparing how well VGG16 and AlexNet performed, we can get a sense of which model might be better suited for different image classification problems.

Conclusion

By using pre-trained models of VGG16 and AlexNet, I was able to compare accuracy.

Problem 5

Exploring feature matching using Histogram of Gradient (HoG) features and Scale-Invariant Feature Transform (SIFT) features. The goal is to match features between an original image and its affine-transformed versions.

Implementation

Corner Detection: We begin by detecting corners in the images using the Shi-Tomasi corner detector. This allows us to identify key points in the images where features can be matched.

Feature Extraction (HoG): Next, we extract HoG features for 8x8 patches around each corner point. HoG features provide information about the local gradient orientation, which is useful for matching similar regions in images.

Matching Features: We match features between the original image and its transformed versions using HoG features. For each feature point, we compute the dot product of feature vectors and consider a match if the dot product exceeds a specified threshold. Normalized cross-correlation is used as the distance measure.

SIFT Feature Extraction: Additionally, we use the SIFT algorithm implemented in the skimage library to extract SIFT features. These features are robust to changes in scale, rotation, and illumination, making them suitable for feature matching.

Conclusion

Feature matching using both HoG and SIFT features provides robust and accurate results.