Task 1:

Task 1: What are the difference between a linear and ~~non le~~ non-linear filter? In which cases does one perform better than the other?

Ans: The diffepences between a linear and non-linear filter are,

	Linear Filters	Non-linear Filters
Operation	Performs a linear operation on each pixel of an image	Performs a non-linear operation on each pixel of an image.
Output	Output is a linear combination of input pixel values, e.g: weighted averaging filter.	Output is not a linear combination of input pixel values, e.g: Median filter.
Properties	Linear filters are commutative, associative, and distributive.	Non-linear filters do not have these properties.
Application	Useful for smoothing, noise reduction, edge detection.	Useful for ~~m~~ removing outliers, preserving edges, contrast enhancement
Examples	Mean filter, Gaussian filter, Laplacian filter	Median filter, Maximum filter, Minimum filter

Non-linear filters performs better than linear filters when the image has complex noise patterns, edges or textures. One of the main advantages of non-linear filters is their ability to preserve image edges while removing noise. Linear filters tend to blur image edges.

Another advantage of non-linear filters is their robustness to outliers. Linear filters are highly sensitive to outliers, and a single outlier can significantly affect the filter output. Which happens when we use linear filter to remove noise, it smears the noise.

~~Non linear filters~~
These are the cases where non-linear filters work better than linear filter.

Task 2

Task 2: What kind of noise is induced in the ~~output~~ input image? How can we achieve the output image?

Ans: Salt and Pepper noise is induced in the input image.

Salt and pepper noise is a type of noise that randomly replaces some pixels in an image with either the minimum or maximum pixel intensity values, which can degrade image quality. In the input image we can see white and black dots scattered in the image. Which indicates the image have salt and pepper noise.

We can ~~the~~ achieve the output image by applying median filter on the input image. The median filter will remove the noise pixels and denoise the input picture.

Code:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('Salt_pepper_2.png')

noise_free = cv2.medianBlur(img,5)
cv2.imwrite("Output_median_blur.png",noise_free)


plt.subplot(1, 2, 1)
plt.xticks([]), plt.yticks([])
plt.imshow(img)
plt.subplot(1, 2, 2)
plt.xticks([]), plt.yticks([])
plt.imshow(noise_free)
plt.show()
```

Result:

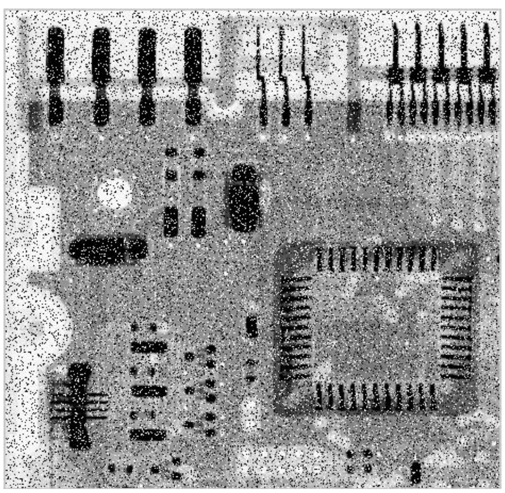

Fig 2.1: Original Image

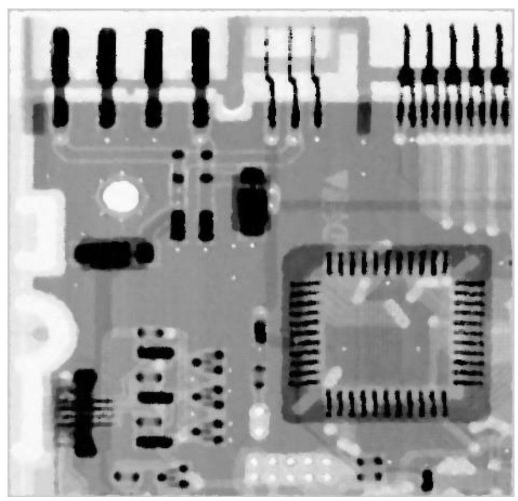

Fig 2.2: Output Image applying Median Filter

Task 3

Task 3: An example of a filter that can perform both edge preservation and noise reduction is the Bilateral filter. This is a non-linear filter used for smoothing images while preserving the edges. It computes the weighted average of neighboring pixels based on both their spatial distance and intensity diffenence. By incorporating both distance and intensity information, the bilateral filter is able to denoise (smooth) ~~the~~ the image while preserving edges. It is most effective reducing Gaussian noise.

Code:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('Gaussian_noise_3.jpg')

noise_free = cv2.bilateralFilter(img,9,75,75)

cv2.imwrite("Output_bilateral_blur_3.png",noise_free)

plt.subplot(1, 2, 1)
plt.xticks([]), plt.yticks([])
plt.imshow(img)
plt.subplot(1, 2, 2)
plt.xticks([]), plt.yticks([])
plt.imshow(noise_free)
plt.show()
```

Result:

Fig 3.1: Input image (with Gaussian Noise).

Fig 3.2: Output Image (Applying Bilateral Filter)

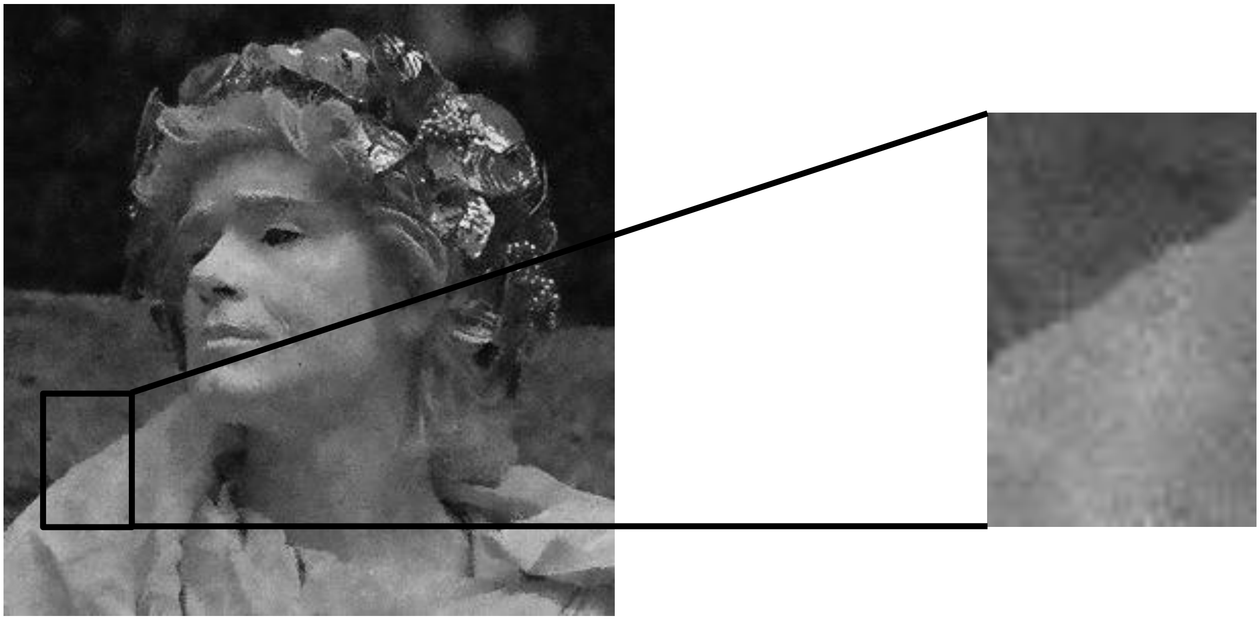

Fig 3.3: Edge Preservation using Bilateral Filter

Task 3:

Another filter that can remove noise while preserving edge values is the Median ~~Fil~~ filter. The median filter replaces each pixel in an image with the median value of its neighboring pixels. Thus it can effectively reduce noise in the image.

In addition, the median filter has a useful property of preserving edges because it does not change the pixel value if the neighboring pixels have similar intensities. This makes the median filter particularly effective in reducing salt and pepper noise. However, median filter ~~is~~ may not be as effective as the bilateral filter in reducing noise while preserving fine details and structure, specially when the noise is Gaussian. So, depending on the task we can choose either the median filter or bilateral filter.

Code:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('salt_and_pepper.png')

noise_free = cv2.medianBlur(img,3)

cv2.imwrite("Output_bilateral_blur_4.png",noise_free)

plt.subplot(1, 2, 1)
plt.xticks([]), plt.yticks([])
plt.imshow(img)
plt.subplot(1, 2, 2)
plt.xticks([]), plt.yticks([])
plt.imshow(noise_free)
plt.show()
```

Result:

Fig 3.4: Input image (with Salt-and-Pepper Noise)

Fig 3.5: Output image (applying Median Filter)

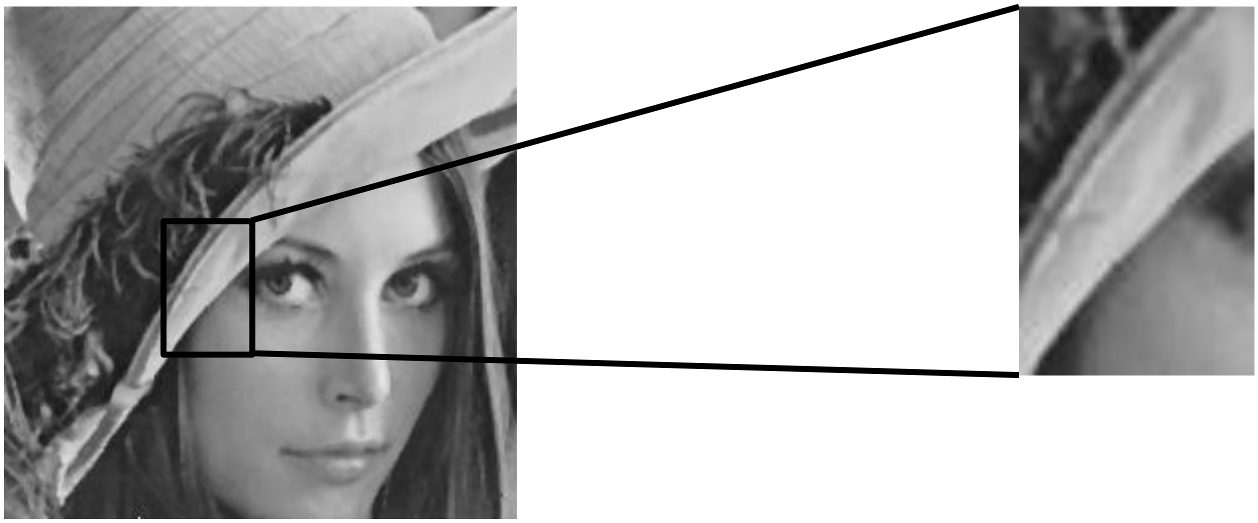

Fig 3.6: Edge Preservation using Median Filter

Task 4

Task 4: We can achieve the kind of output given in "4.png" by ~~using~~ image blending or image compositing. The basic idea is to use a mask to control how much of each input image is included in the final image. This mask can be thought of as a transparency layer that determines which parts of each input image are visible in the final image.

Code:

```
import cv2
import numpy as np

# Load the two images and the mask
image1 = cv2.imread('ironman.png')
image1 = np.asarray(image1, np.uint8)
image2 = cv2.imread('hulk.png')
image2 = np.asarray(image2, np.uint8)
mask = cv2.imread('mask.png')
mask = np.asarray(mask, np.uint8)

height, width, channels = image1.shape
print(height, width)
image2 = cv2.resize(image2,(width, height))
mask = cv2.resize(mask, (width, height))

# Normalize the mask to be between 0 and 1
mask = mask / 255
mask = np.asarray(mask, np.uint8)
print(mask)

# Multiply the images by the mask
image1_masked = cv2.multiply(image1, mask)

image2_masked = cv2.multiply(image2, 1 - mask)

# Add the masked images together
combined_image = cv2.add(image1_masked, image2_masked)

# Display the result
cv2.imwrite("Ironman_Hulk.png",combined_image)
cv2.imshow('Combined Image', combined_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result:

Fig 4.1: Input image 1

Fig 4.2: Input image 2

Fig 4.4: Combined Image (A skinnier Hulk with Ironman Helmet)