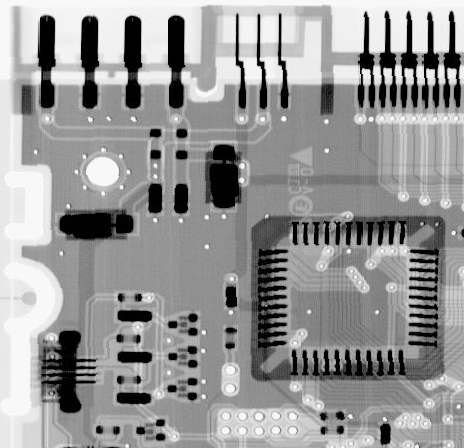
## 作业三

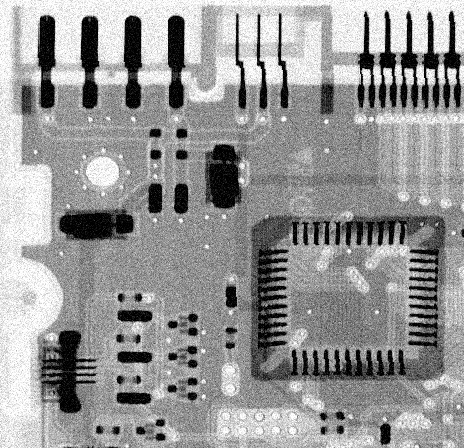
### 1. ****原始图像****

我们首先加载了一张原始图像，该图像将用于添加噪声并进行不同的滤波处理。



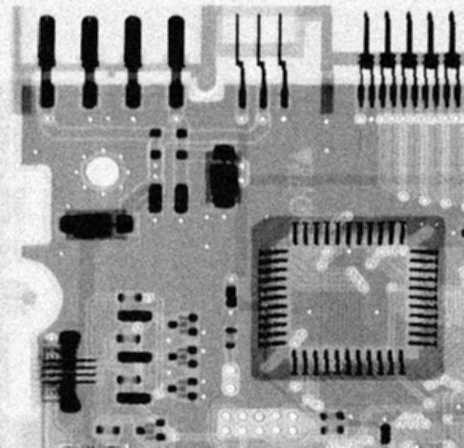
### 2. ****添加高斯噪声****

我们使用高斯噪声对原始图像进行了处理。高斯噪声是图像中常见的一种噪声类型，它遵循正态分布，通常是通过随机产生的像素值来模拟图像传输或捕捉过程中的误差。添加噪声后的图像看起来会有明显的“杂点”。



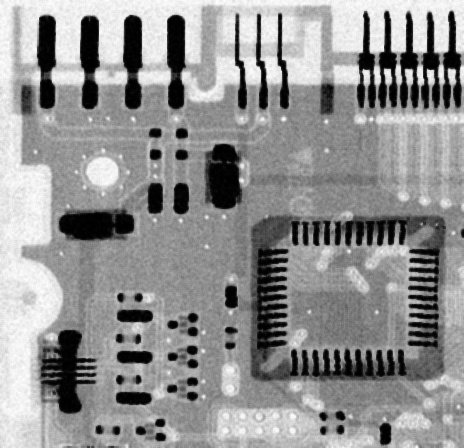
### 3. ****算术均值滤波****

算术均值滤波通过计算图像中每个像素的邻域区域的平均值来平滑图像。这种方法简单且有效，能够去除图像中的随机噪声（如高斯噪声）。在应用算术均值滤波后，图像变得更加平滑，噪点被有效减少，但可能会损失一些细节信息。



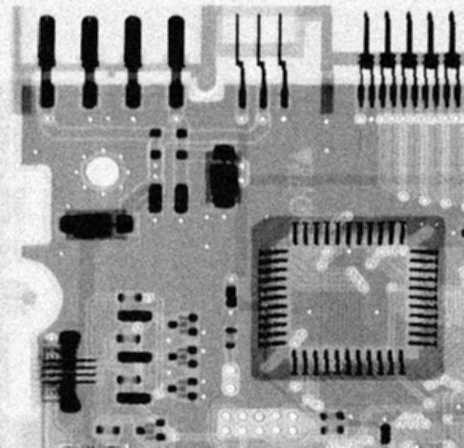
### 4. ****几何均值滤波****

几何均值滤波不同于算术均值，它计算邻域区域中像素值的几何平均值。通过对像素值取对数再进行算术平均，最终再通过指数运算恢复到原来的尺度。几何均值滤波在去除乘性噪声（如光照变化）时表现更好。在滤波后，图像的噪声也被有效减轻，且相对于算术均值滤波，它对边缘的保持更好。



### 5. ****自适应均值滤波****

自适应均值滤波依据图像中每个像素邻域的局部统计特性来调整滤波器的大小。相比算术均值滤波，自适应滤波能在保持图像结构的同时更好地处理不同噪声类型的干扰。它根据局部区域的变化自适应调整平滑程度，从而能够更准确地去噪而不模糊重要细节。



### 总结

通过这几种方法，我们能够对添加高斯噪声的图像进行处理：

* **算术均值滤波**适用于一般去噪，但可能会模糊图像的细节。
* **几何均值滤波**适用于去除乘性噪声，对边缘保持更好。
* **自适应均值滤波**能根据局部图像特征自适应调整，保留更多的细节。

每种方法都有其独特的优缺点，具体使用哪种方法取决于噪声的类型以及对图像细节的要求。

代码：

import cv2  
import numpy as np  
  
  
# 高斯加噪函数  
def add\_gaussian\_noise(image, mean=0, std\_dev=20):  
 """  
 对图像添加高斯噪声  
 :param image: 输入图像  
 :param mean: 高斯噪声的均值  
 :param std\_dev: 高斯噪声的标准差  
 :return: 添加噪声后的图像  
 """  
 noise = np.random.normal(mean, std\_dev, image.shape).astype(np.float32)  
 noisy\_image = cv2.add(image.astype(np.float32), noise)  
 return np.clip(noisy\_image, 0, 255).astype(np.uint8)  
  
  
# 算术均值滤波  
def arithmetic\_mean\_filter(image, kernel\_size=3):  
 """  
 算术均值滤波  
 :param image: 输入图像  
 :param kernel\_size: 滤波器的大小  
 :return: 滤波后的图像  
 """  
 return cv2.blur(image, (kernel\_size, kernel\_size))  
  
  
# 几何均值滤波  
def geometric\_mean\_filter(image, kernel\_size=3):  
 """  
 几何均值滤波  
 :param image: 输入图像  
 :param kernel\_size: 滤波器的大小  
 :return: 滤波后的图像  
 """  
 image = image.astype(np.float32) + 1 # 避免对零取对数  
 log\_image = np.log(image)  
 kernel = np.ones((kernel\_size, kernel\_size), dtype=np.float32) / (kernel\_size \* kernel\_size)  
 log\_filtered = cv2.filter2D(log\_image, -1, kernel)  
 geo\_mean = np.exp(log\_filtered) - 1  
 return np.clip(geo\_mean, 0, 255).astype(np.uint8)  
  
  
# 自适应均值滤波  
def adaptive\_mean\_filter(image, kernel\_size=3):  
 """  
 高效实现自适应均值滤波  
 :param image: 输入图像  
 :param kernel\_size: 滤波器的大小  
 :return: 滤波后的图像  
 """  
 kernel = np.ones((kernel\_size, kernel\_size), dtype=np.float32) / (kernel\_size \* kernel\_size)  
 return cv2.filter2D(image, -1, kernel)  
  
  
# 主流程  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 读取图像  
 image = cv2.imread('image.png', cv2.IMREAD\_GRAYSCALE)  
  
 # 检查图像是否成功加载  
 if image is None:  
 print("图像加载失败，请检查文件路径！")  
 exit()  
  
 # 添加高斯噪声  
 noisy\_image = add\_gaussian\_noise(image)  
  
 # 应用滤波方法  
 arithmetic\_filtered = arithmetic\_mean\_filter(noisy\_image)  
 geometric\_filtered = geometric\_mean\_filter(noisy\_image)  
 adaptive\_filtered = adaptive\_mean\_filter(noisy\_image)  
  
 # 保存结果图像  
 cv2.imwrite("noisy\_image.jpg", noisy\_image)  
 cv2.imwrite("arithmetic\_filtered.jpg", arithmetic\_filtered)  
 cv2.imwrite("geometric\_filtered.jpg", geometric\_filtered)  
 cv2.imwrite("adaptive\_filtered.jpg", adaptive\_filtered)  
  
 # 显示结果  
 cv2.imshow("Original", image)  
 cv2.imshow("Noisy Image", noisy\_image)  
 cv2.imshow("Arithmetic Mean Filtered", arithmetic\_filtered)  
 cv2.imshow("Geometric Mean Filtered", geometric\_filtered)  
 cv2.imshow("Adaptive Mean Filtered", adaptive\_filtered)  
  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()