Project TumorTrace: MRI-Based AI for Breast Cancer Detection



Infosys SPRINGBOARD 5.0

Name : Suyash Panchal Company Name : Infosys Role : AI/ML Intern

Duration : 7th October 2024 – December 2024 Linkedln : <https://www.linkedin.com/in/suyash-panchal-4970b530a>

Github : <https://github.com/Suyashpanchal07/TumorTrace.git>

AI/ML Internship Assignments and Tasks

**Day-1**

8-10-2024

# Introduction to TumorTrace Project

Project TumorTrace focuses on using artificial intelligence (AI) to enhance breast cancer detection through MRI scans.

# Project dataset and basic concepts

Dataset used : <https://drive.google.com/file/d/1rKps09z1DEkfiICZlpIBOFek-LZ1vFAC/view>

The project aims to:

Improve Early Detection: Develop an AI model that analyzes MRI images to identify breast cancer at earlier stages than traditional methods.

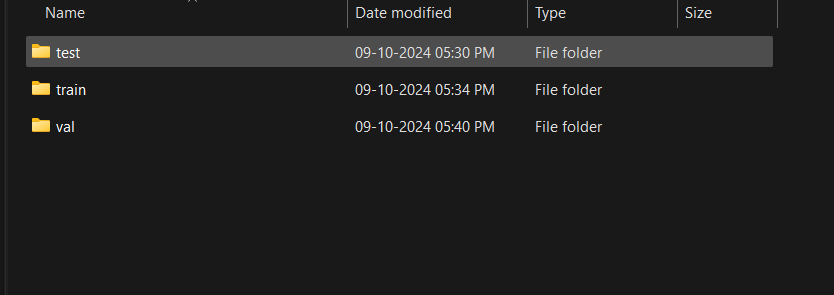
Utilize Deep Learning: Implement deep learning algorithms, such as convolutional neural networks (CNNs), for accurate feature extraction and classification of tumors.

Validate Performance: Train and validate the model using a diverse dataset of MRI images, ensuring high accuracy and reliability.

Clinical Integration: Collaborate with healthcare professionals to create user-friendly tools for radiologists, facilitating seamless integration into clinical workflows.

Enhance Patient Outcomes: Ultimately, the project seeks to improve patient outcomes through earlier diagnosis and personalized treatment plans.

Dataset files:



**Day-2**

9-10-2024

# Setting up GitHub and Python Environments

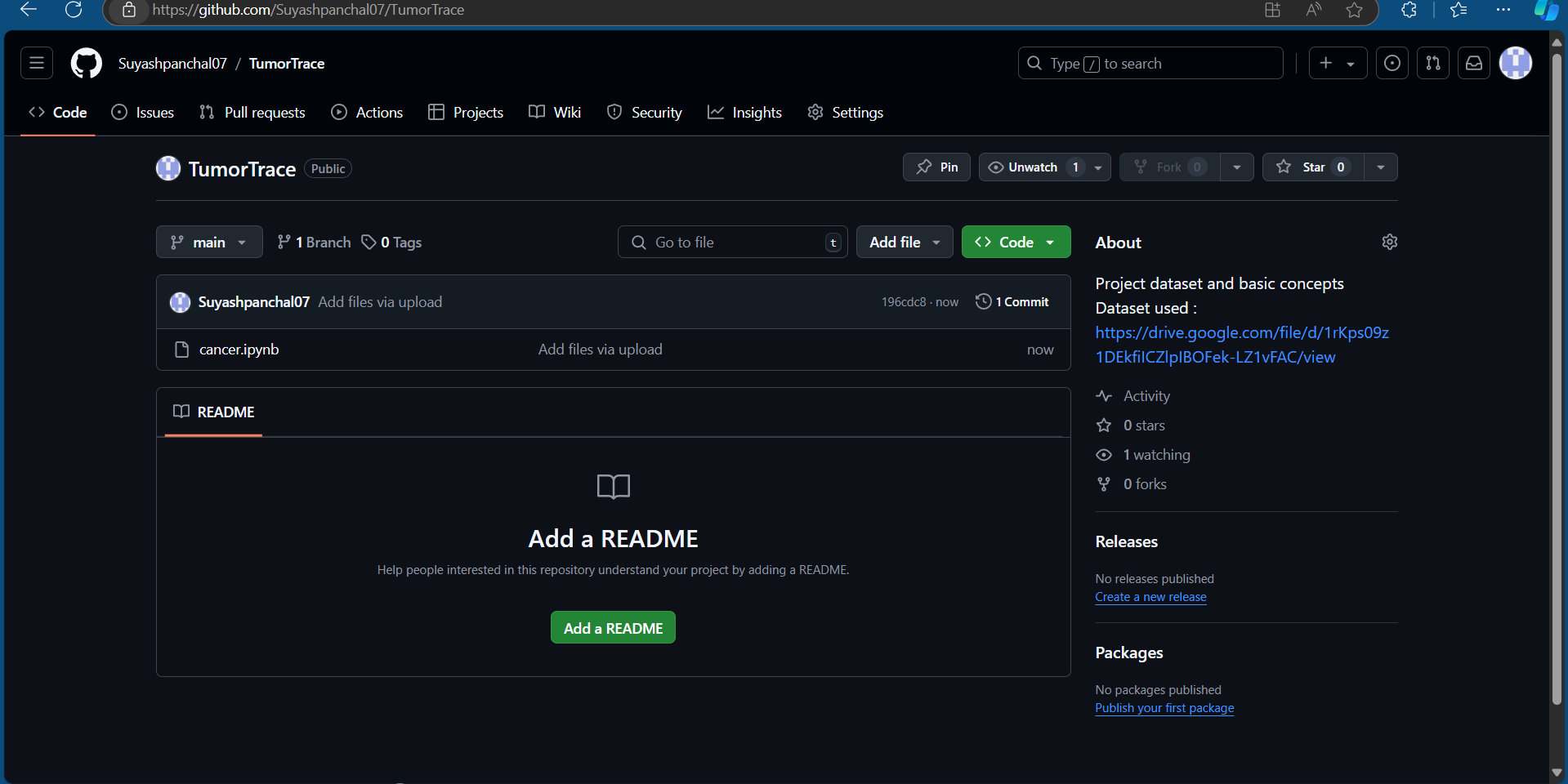
Step 1: Install Prerequisites

1. Install Git from the official website.
2. Install Python from the official website, ensuring to add it to your system PATH. (installed version - Python 3.11.4)
3. Install Visual Studio Code from the official website. Step 2: Set Up GitHub Repository

1. Create a new repository on GitHub, naming it (e.g., TumorTrace) and setting the desired visibility.

Step 3: Clone the Repository

1. Open the terminal in Visual Studio Code.
2. Clone your GitHub repository to your local machine. Step 4: Set Up Python Environment
3. Navigate to your project directory in the terminal.
4. Create a virtual environment for your project.
5. Activate the virtual environment.
6. Install the required Python packages for your project. Step 5: Open the Project in VS Code
7. Open your project folder in Visual Studio Code.
8. Select the Python interpreter associated with your virtual environment. Step 6: Start Coding
9. Create new files for your project - Data.ipynb
10. Use Git to track changes and commit your work regularly.



**Day-3**

10-10-2024

# Understanding CNN architecture

Reference site - <https://poloclub.github.io/cnn-explainer/>

Brief Summary of the Content:

The CNN Explainer is an interactive tool designed to help users understand the fundamental concepts behind Convolutional Neural Networks (CNNs), which are commonly used for image processing tasks in machine learning. Here’s a summary of its main features and content:

**Key Concepts**

1. **Convolutional Neural Networks (CNNs)**:
   * CNNs are a type of deep learning model that excels in processing and analyzing visual data, such as images. They are composed of several layers, each responsible for extracting different levels of features from the input image.
2. **Interactive Visualization**:
   * The tool provides a visual, hands-on way to learn about CNNs by showing how input images are processed step-by-step through different layers of a network. Users can interact with each component to see how the model extracts features, transforms data, and makes predictions.
3. **Layer Explanation**:
   * **Convolutional Layer**: Explains how the convolution operation extracts features from the image using filters. It shows how different filters can detect edges, textures, and patterns.
   * **Activation Layer**: Demonstrates the role of activation functions (like ReLU) in introducing non-linearity to the network, allowing it to learn complex patterns.
   * **Pooling Layer**: Describes how pooling (e.g., max-pooling) reduces the spatial dimensions of the feature maps, retaining important features while reducing computation.
   * **Fully Connected Layer**: Shows how the final layers take the extracted features and use them to classify images into different categories.
4. **Real-Time Examples**:
   * Users can upload their own images or use sample images provided to see how a CNN processes real-world inputs. This helps to understand how different layers contribute to the final output, which could be an object classification or identification.
5. **Educational Insights**:
   * The explainer offers educational tips, definitions, and simple explanations of complex concepts, making it easier for beginners to grasp how CNNs work. It covers terms like feature maps, kernels, and stride, and explains their significance in the network's functioning.



**Day-4**

11-10-2024

# Performing Exploratory Data Analysis

**Transformation:** The transform in PyTorch is set to resize the images to (224, 224) and convert them to tensors.

**Dataset Loading:** Used datasets.ImageFolder to load images, which assumes a directory structure where each subdirectory is a class.

**DataLoader:** Creates batches and shuffles the dataset.

**Image Display:** Adjusted the method to filter and retrieve images of a specific class and converted tensors to NumPy arrays for displaying with matplotlib.

import torch

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

# Define transformation (resize and convert to tensor)

transform = transforms.Compose([

transforms.Resize((224, 224)), # Resize all images to 224x224

transforms.ToTensor() # Convert images to PyTorch tensors

])

# Load the dataset from a directory

dataset = datasets.ImageFolder(D:\\Infosys Internship\\test', transform=transform)

# Create a data loader without splitting

data\_loader = DataLoader(dataset, batch\_size=1, shuffle=True)

# Iterate over a batch of images

for images, labels in data\_loader:

# Convert tensor to numpy for displaying

img = images[0].permute(1, 2, 0).numpy() # Change dimensions for plotting

plt.figure(figsize=(5, 5))

plt.imshow(img)

plt.title(f'Label: {labels[0].item()}')

plt.axis('off')

plt.show()

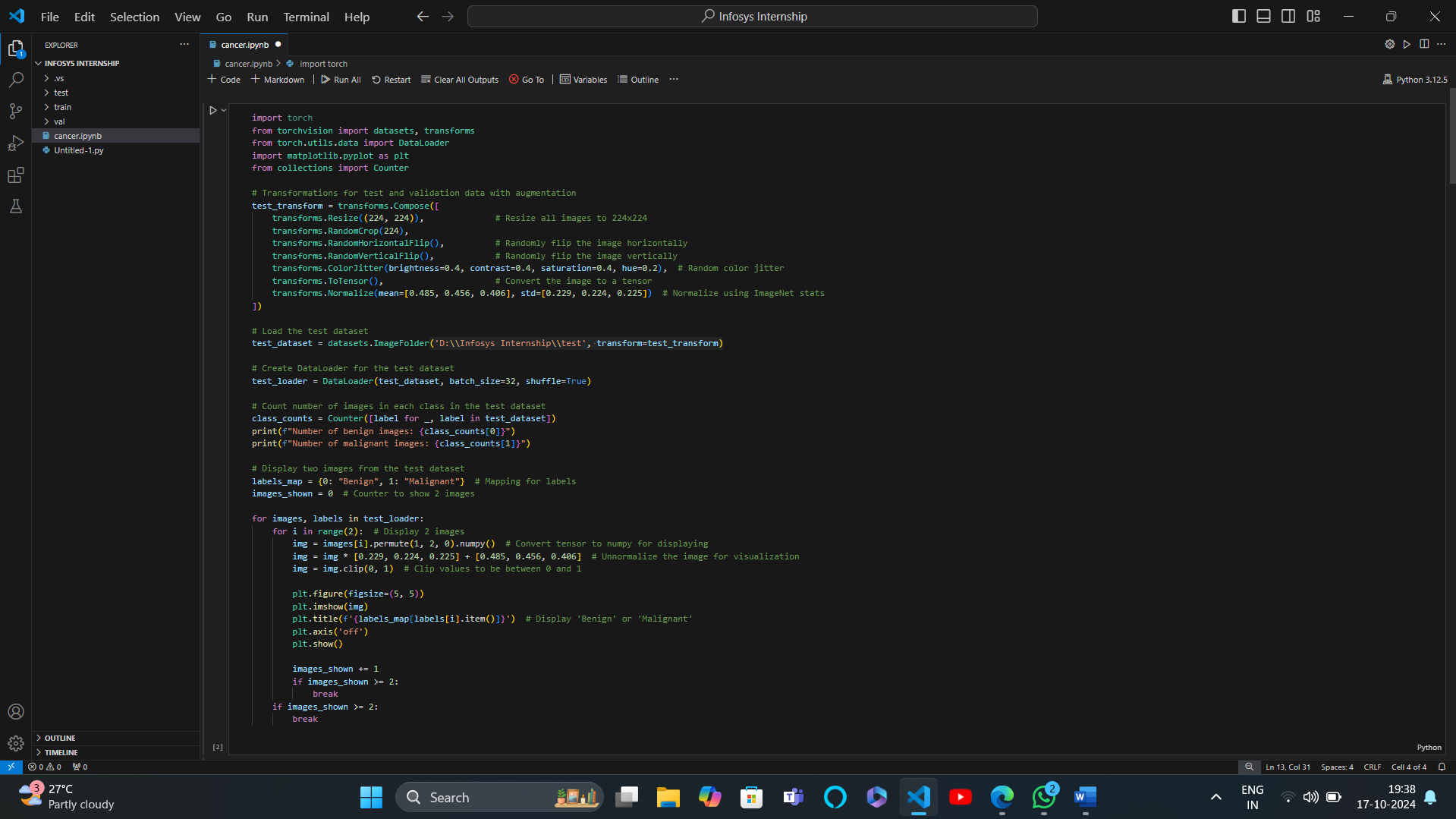
break # Display only one image

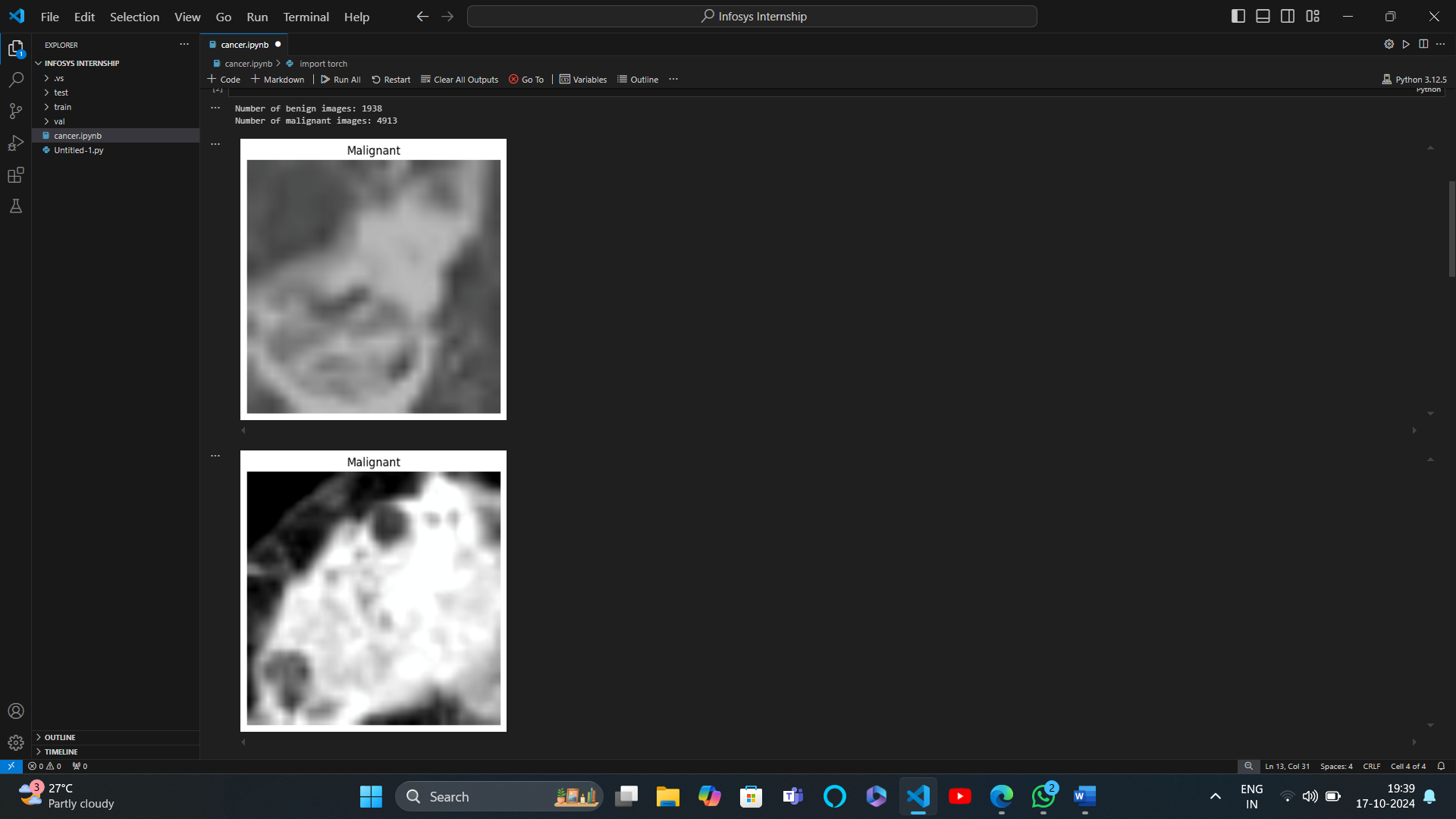
## Implementing data augmentation techniques

**Data Augmentation:** The code applies various transformations (resizing, flipping, rotation, zooming, and color adjustments) to augment the training images, enhancing dataset diversity for better model training.

**Dataset Handling:** It utilizes torchvision.datasets.ImageFolder to load images organized by class from a specified directory, along with a DataLoader for efficient batch processing and shuffling.

**Visualization**: A function is defined to display a specified number of augmented images, allowing for visual inspection of the transformations applied, which aids in understanding the data variations presented to the model.





**Day-5**

14-10-2024

# Understanding ResNet and VGG16 architecture

Reference: [Understanding and visualizing ResNets | by Pablo Ruiz | Towards Data Science](https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8)  
 [Everything you need to know about VGG16 | by Great Learning | Medium](https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918)

ResNet (Residual Networks)

Overview: ResNet, or Residual Network, is a type of deep convolutional neural network (CNN) architecture designed to facilitate the training of very deep networks. Introduced by Kaiming He and his colleagues in their 2015 paper "Deep Residual Learning for Image Recognition," ResNet has become a foundational model in computer vision tasks.

Key Features:

Residual Learning:

ResNet introduces the concept of residual learning, where the network learns to predict the residual (the difference) between the input and the output of a series of layers. This is achieved through skip connections that allow the input to bypass one or more layers.

Skip Connections:

Skip connections (or identity mappings) help mitigate the vanishing gradient problem, which often occurs when training deep networks. By allowing gradients to flow directly through these connections, ResNet can train effectively even with hundreds or thousands of layers.

Architecture:

A typical ResNet architecture consists of multiple residual blocks, each containing two or three convolutional layers, followed by batch normalization and ReLU activation functions. The output of each block is added to the input (the shortcut connection), which enables the network to learn identity functions when needed.

Depth:

ResNet architectures can vary in depth, with common configurations including ResNet- 18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, indicating the number of layers in the network. The deeper networks often achieve higher accuracy on complex tasks.

Performance:

ResNet has demonstrated remarkable performance in image classification tasks, notably winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2015. It has since been widely adopted in various applications, including object detection, segmentation, and more.

Generalization:

The architecture's ability to train deeper networks without overfitting makes it suitable for a wide range of applications, helping to improve generalization on unseen data.

## Impact and Applications

State-of-the-Art: ResNet set new benchmarks in several computer vision challenges and has inspired a wide range of subsequent architectures, including DenseNet and EfficientNet.

Transfer Learning: Pre-trained ResNet models are commonly used in transfer learning scenarios, allowing practitioners to leverage the power of deep learning even with limited data for specific tasks.

Broader Use: Beyond image classification, ResNet has been adapted for tasks in natural language processing, audio recognition, and more, showcasing its versatility.

## Conclusion

ResNet represents a significant advancement in the field of deep learning, particularly in training deep neural networks. Its innovative use of residual learning and skip connections has not only improved performance on various tasks but also laid the groundwork for future research and architectural innovations in the field.

VGG16 is a convolutional neural network architecture that was developed by the Visual Graphics Group (VGG) at the University of Oxford. It is widely used in the field of computer vision for image classification and feature extraction. Here are some key points about the VGG16 model:

1. Architecture
   * Depth: VGG16 is a deep neural network with 16 layers that have learnable weights, hence the name VGG16. The architecture consists of:
     + 13 convolutional layers
     + 3 fully connected layers
     + 5 max-pooling layers
   * Convolutional Layers: The convolutional layers use small 3x3 filters with a stride of
2. This small kernel size helps the network learn complex features while maintaining spatial resolution.
   * Pooling Layers: Max-pooling layers with a 2x2 filter and a stride of 2 are used to down-sample feature maps, reducing the spatial dimensions while retaining important information.
3. Input Size
   * VGG16 accepts input images of size 224x224 pixels. Images are typically preprocessed by subtracting the mean RGB value, calculated from the training set.
4. Activation Function
   * The ReLU (Rectified Linear Unit) activation function is used after each convolutional layer to introduce non-linearity and help the model learn complex patterns.
5. Fully Connected Layers
   * The fully connected layers at the end of the network consist of 4096, 4096, and 1000 units, respectively. The final layer uses the softmax activation function to output class probabilities for 1000 classes in the ImageNet dataset.
6. Training
   * VGG16 was initially trained on the ImageNet dataset, which contains millions of images across thousands of classes. The model achieved significant performance, winning the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) in 2014.
7. Transfer Learning
   * VGG16 is often used for transfer learning due to its ability to extract features from images. Pre-trained VGG16 weights can be fine-tuned for specific tasks, allowing users to leverage the model's learned representations without requiring a large dataset.
8. Applications
   * VGG16 is used in various computer vision tasks, including:
     + Image classification
     + Object detection
     + Image segmentation
     + Feature extraction for other models
9. Limitations
   * While VGG16 is known for its simplicity and effectiveness, it has a large number of parameters (approximately 138 million), making it computationally intensive. This can lead to long training times and increased memory requirements.

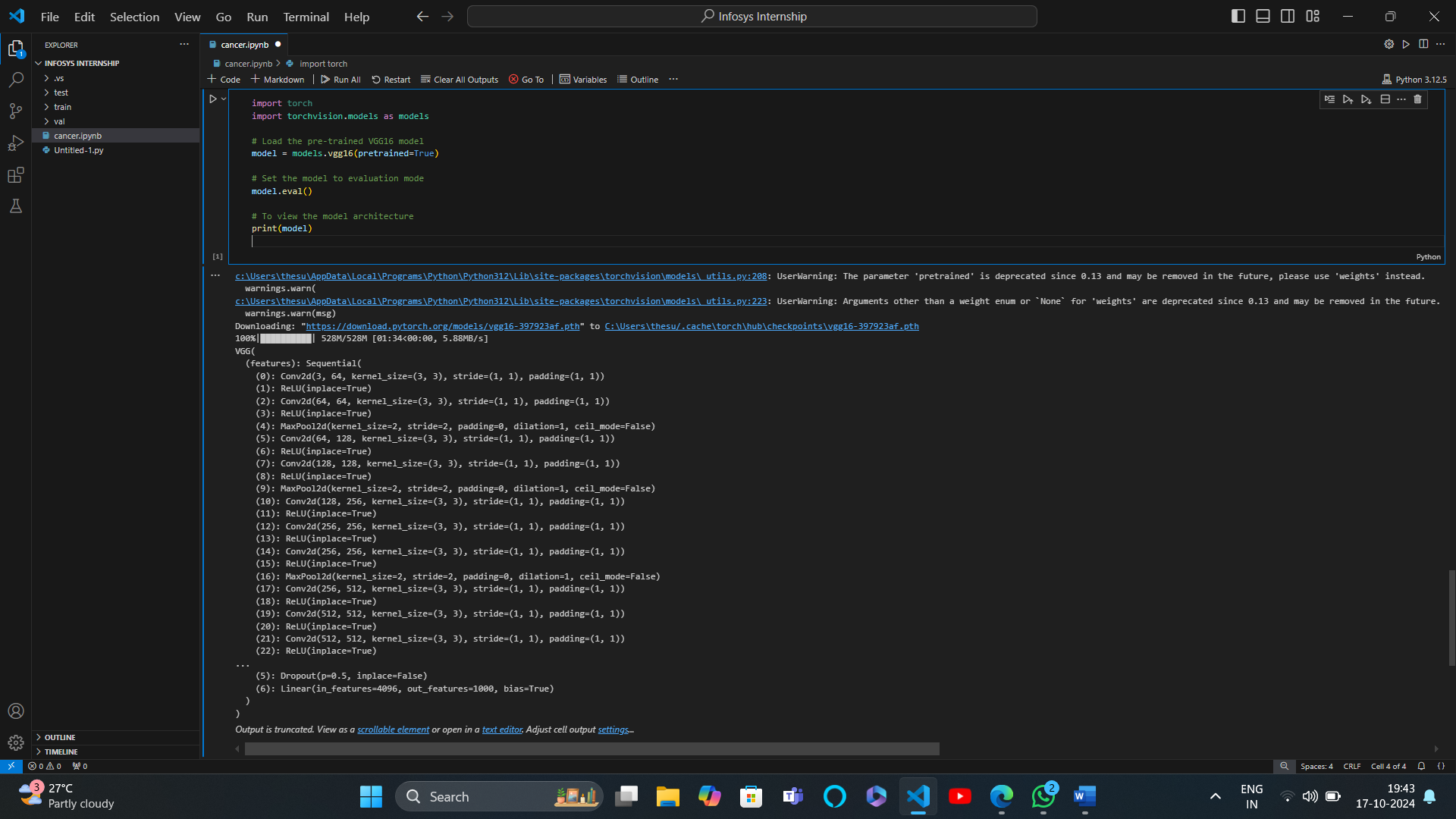
Conclusion

VGG16 is a foundational model in deep learning and computer vision. Its architecture, characterized by small convolutional filters and deep layers, has influenced many subsequent models and remains a popular choice for image-related tasks, especially in the context of transfer learning.

**Day-6**

15-10-2024

Load a VGG16 Model



## Why Convolutional Neural Networks (CNNs) are generally more efficient than fully connected networks for image classification ??

Reference: [Fully Connected Layer vs Convolutional Layer - GeeksforGeeks](https://www.geeksforgeeks.org/fully-connected-layer-vs-convolutional-layer/)

1. Parameter Sharing

Weight Sharing: In CNNs, the same convolutional filter (kernel) is applied across the entire image. This means that the same set of weights is used to detect features at different spatial locations, significantly reducing the number of parameters compared to fully connected networks, where each input is connected to every neuron in the next layer.

1. Local Connectivity

Localized Feature Detection: CNNs utilize local receptive fields, meaning each neuron in a convolutional layer is connected only to a small region of the input image. This allows CNNs to learn spatial hierarchies of features, such as edges, textures, and shapes, which are crucial for image classification.

1. Spatial Hierarchy of Features

Layered Feature Learning: CNNs can learn features at multiple levels of abstraction. Early layers may learn simple features (like edges), while deeper layers can combine these features to recognize more complex patterns (like shapes and objects). This hierarchical representation is particularly effective for images.

1. Reduced Overfitting

Fewer Parameters: With fewer parameters due to weight sharing and local connectivity, CNNs are less prone to overfitting, especially when trained on smaller datasets. This allows them to generalize better to unseen data.

1. Translation Invariance

Feature Detection: CNNs exhibit some degree of translation invariance because the same filters are applied across the entire image. This means that objects in different positions can still be recognized effectively, unlike fully connected networks, which do not account for the spatial arrangement of features.

1. Pooling Layers

Dimensionality Reduction: CNNs typically incorporate pooling layers (like max pooling) to down- sample feature maps. This not only reduces the spatial dimensions of the data, making computations more efficient, but also helps in achieving invariance to small translations and distortions.

1. Computational Efficiency

Convolution Operations: The mathematical operations in CNNs, particularly convolution, can be computed efficiently using specialized hardware (like GPUs) and optimized algorithms. This allows CNNs to process large images quickly compared to fully connected networks, which involve more computationally intensive matrix multiplications.

1. Transfer Learning

Pre-trained Models: CNN architectures are often pre-trained on large datasets (like ImageNet), allowing them to leverage learned features for specific tasks. This makes them more efficient for image classification tasks, as they can achieve high performance even with limited training data.

Conclusion

Overall, CNNs are designed to exploit the spatial structure and patterns within images, making them more effective and efficient for image classification tasks compared to fully connected networks. Their architecture allows for better feature extraction, reduced computational requirements, and improved generalization, which are critical factors in achieving high accuracy in image-related tasks. Feature extraction is a crucial process in machine learning and computer vision that involves transforming raw data into a set of measurable characteristics (features) that can be used for analysis, classification, or prediction.

## ABOUT FEATURE EXTRACTION

1. Definition

Feature extraction refers to the process of identifying and isolating the relevant information or attributes from raw data. In the context of images, audio, or text, features are the characteristics that represent the essential content of the data.

1. Importance

Dimensionality Reduction: By focusing on key features, the complexity of the data is reduced, making models easier to train and less prone to overfitting.

Improved Performance: Well-chosen features can significantly enhance the performance of machine learning algorithms by providing them with relevant information.

Interpretability: Features can make it easier to understand the decision-making process of models, allowing for better insights into the underlying data.

1. Common Techniques
   1. Image Feature Extraction

Edge Detection: Techniques like the Sobel operator, Canny edge detector, or Laplacian filter identify the edges in an image, which can be important features for object detection.

Texture Features: Methods like Local Binary Patterns (LBP) or Gabor filters analyze the texture of images, which can help in identifying patterns and objects.

Histogram of Oriented Gradients (HOG): This technique captures the distribution of directions of gradients, useful for detecting objects like pedestrians.

Convolutional Neural Networks (CNNs): CNNs automatically learn hierarchical features from raw pixel data. Different layers in CNNs extract features at varying levels of abstraction, from edges and textures to shapes and objects.

* 1. Text Feature Extraction

Bag of Words (BoW): This approach represents text data by counting the occurrence of each word in the document, disregarding grammar and order.

Term Frequency-Inverse Document Frequency (TF-IDF): This method weighs the importance of words in documents by considering their frequency in a specific document relative to their frequency in a corpus.

Word Embeddings: Techniques like Word2Vec or GloVe convert words into dense vectors that capture semantic meaning, allowing models to understand the relationships between words.

N-grams: This method involves creating sequences of n words (or characters) to capture context and local patterns in the text.

* 1. Audio Feature Extraction

Mel-Frequency Cepstral Coefficients (MFCCs): This technique is widely used in speech and audio processing to represent the power spectrum of sound.

Spectrograms: Visual representations of the spectrum of frequencies in sound, useful for analyzing audio signals and extracting features related to pitch and tone.

Zero-Crossing Rate: This measures the rate at which the audio signal changes from positive to negative or vice versa, providing information about the signal's noisiness.

1. Applications

Computer Vision: Object recognition, facial recognition, and image classification rely on effective feature extraction.

Natural Language Processing (NLP): Sentiment analysis, topic modeling, and machine translation depend on extracting meaningful features from text data.

Speech Recognition: Feature extraction techniques are essential for converting audio signals into text.

1. Challenges

Feature Selection: Identifying the most relevant features from a potentially vast set can be difficult. Irrelevant or redundant features can lead to poorer model performance.

Feature Engineering: The process of manually creating features can be time-consuming and requires domain expertise.

Overfitting: Including too many features can lead to models that perform well on training data but poorly on unseen data.

Conclusion

Feature extraction is a foundational step in data preprocessing that enhances the effectiveness of machine learning models. By transforming raw data into meaningful features, it enables algorithms to learn more efficiently and accurately, paving the way for advancements in various applications, from image recognition to natural language processing.

Total number of parameters

The total number of parameters in the VGG16 model is \*\*140,095,656\*\*. This large number of parameters contributes to the model's ability to learn complex features but also makes it computationally intensive and prone to overfitting if not managed properly.

**Day-7**

16-10-2024

**Histogram of Oriented Gradients (HOG)** is a feature descriptor used in computer vision and image processing, primarily for object detection. It captures the structure or the shape of an object by analyzing the gradient directions (orientations) in localized parts of an image. Here's a breakdown of how HOG works:

1. **Key Concepts**
   * **Gradient:** Measures the change in pixel intensity (brightness) in the image. It helps to detect edges and texture patterns.
   * **Orientation:** The angle of the gradient. It shows the direction of the edges.
   * **Histogram:** A representation of the distribution of gradients (how often each gradient direction appears).

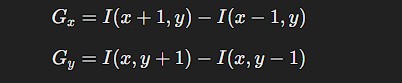
### Steps to Compute HOG

#### Step 1: Preprocessing

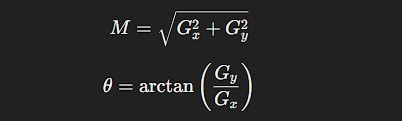
* + The image can be resized to a standard size and, in some cases, converted to grayscale.
  + Normalization can be applied to reduce the effects of lighting conditions.

#### Step 2: Compute Gradients

* + The gradients of the image are computed using techniques like the Sobel operator.
  + Gradients are calculated in both the horizontal (GxG\_xGx) and vertical (GyG\_yGy) directions



* + **Magnitude** (MMM) and **orientation** (θ\thetaθ) of the gradient are calculated as:



#### Step 3: Divide Image into Cells

* + The image is divided into small, square blocks (e.g., 8x8 or 16x16 pixels), called

#### cells.

* + Gradients are calculated for each pixel in the cell, and a histogram of gradient directions (orientations) is created.

#### Step 4: Create Histograms

* + For each cell, a histogram of gradient directions is formed by quantizing the orientations into a fixed number of bins (e.g., 9 bins for 0° to 180°).
  + Each pixel votes for a bin based on its gradient magnitude and orientation.

#### Step 5: Block Normalization

* + To make the HOG descriptor invariant to lighting changes and shadows, several neighboring cells are grouped into a larger unit called a **block** (e.g., 2x2 cells).
  + The histograms in the block are then normalized to ensure that the feature descriptor is not sensitive to variations in lighting.

#### Step 6: Create the HOG Descriptor

* + The normalized histograms from all blocks are concatenated into a single feature vector, which serves as the final HOG descriptor for the image.
  + This feature vector can then be used for machine learning tasks like classification or object detection.

### Advantages of HOG

* + **Robustness:** It is robust to variations in lighting and pose, making it useful for tasks like pedestrian detection.
  + **Captures Local Features:** Effectively captures local edge structures, which are important for recognizing shapes.
  + **Efficient Computation:** Computationally efficient and works well for real-time applications.

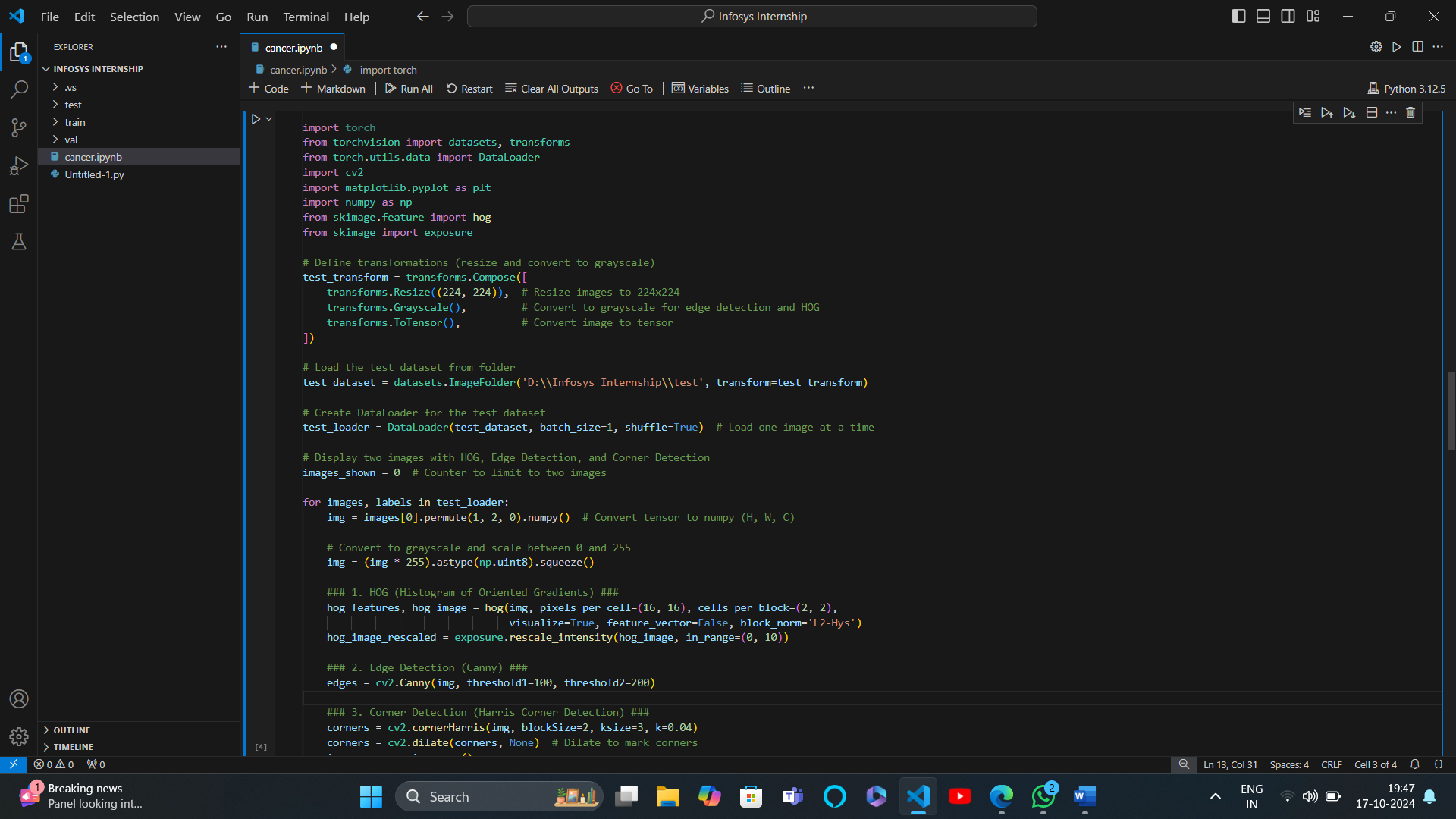
### Applications of HOG

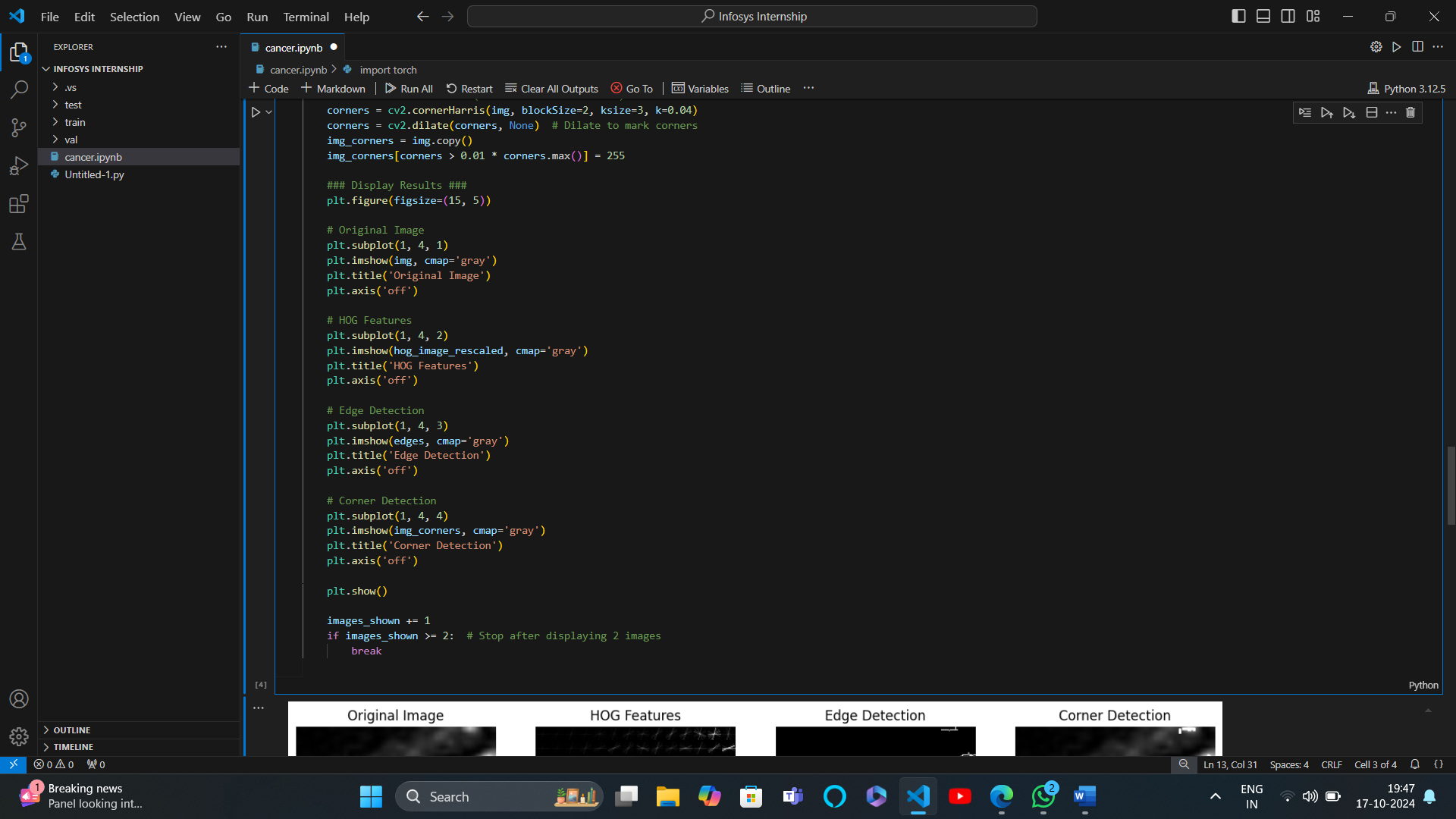
* + **Object Detection:** Widely used for pedestrian detection in images and videos.
  + **Face Recognition:** Can be used to extract features from faces, aiding in recognition systems.
  + **Image Classification:** Serves as a preprocessing step for machine learning models, allowing them to learn meaningful patterns.

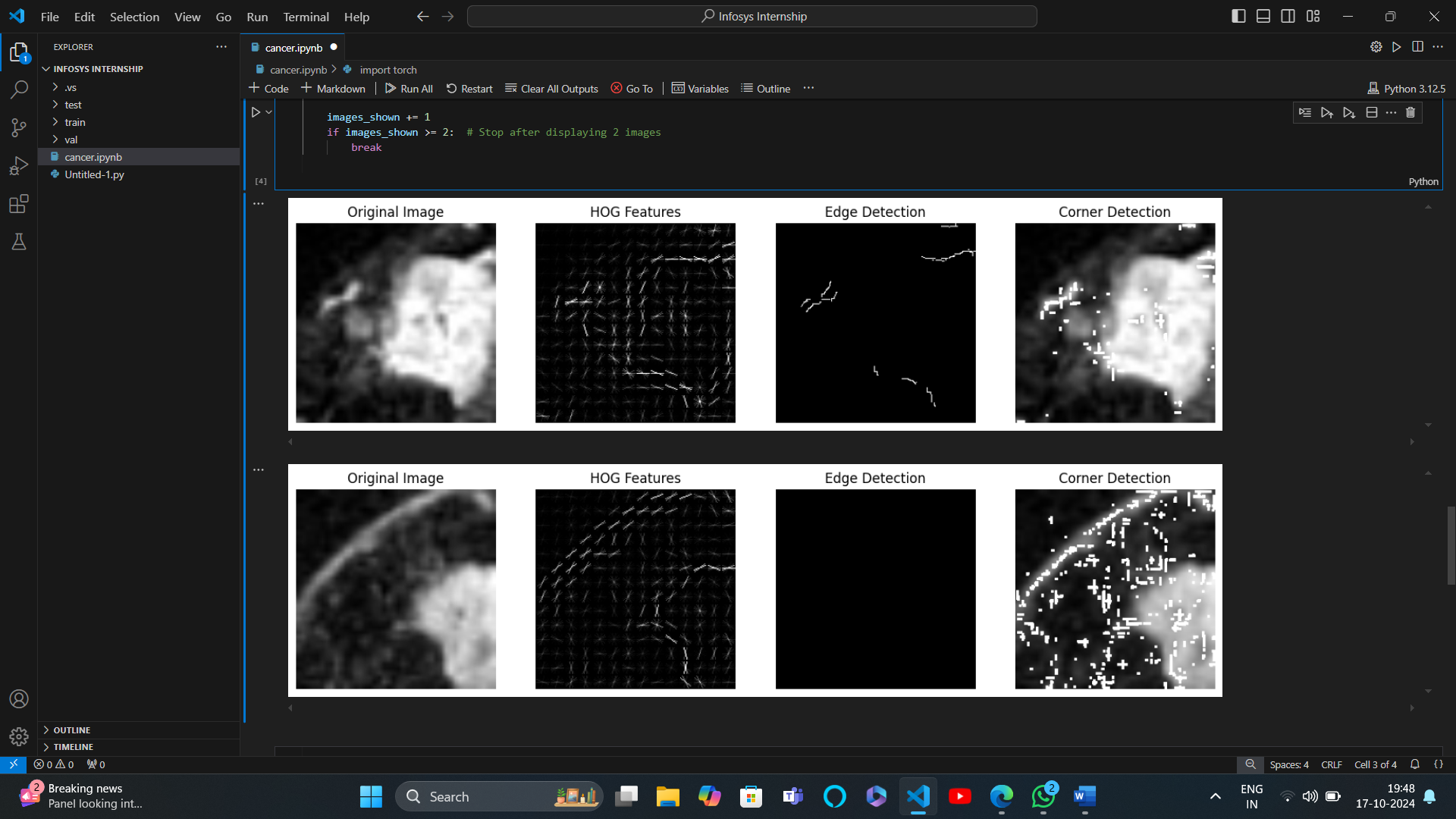
### Visualization

To visualize HOG, it’s common to represent the feature descriptor by drawing lines that show the dominant gradient directions in each cell, providing insight into the edges and structures the descriptor has captured. This helps understand which features are contributing most to object detection.

## IMPLEMENT HISTOGRAM OF ORIENTED GRADIENTS ,CORNER AND EDGE DETECTION



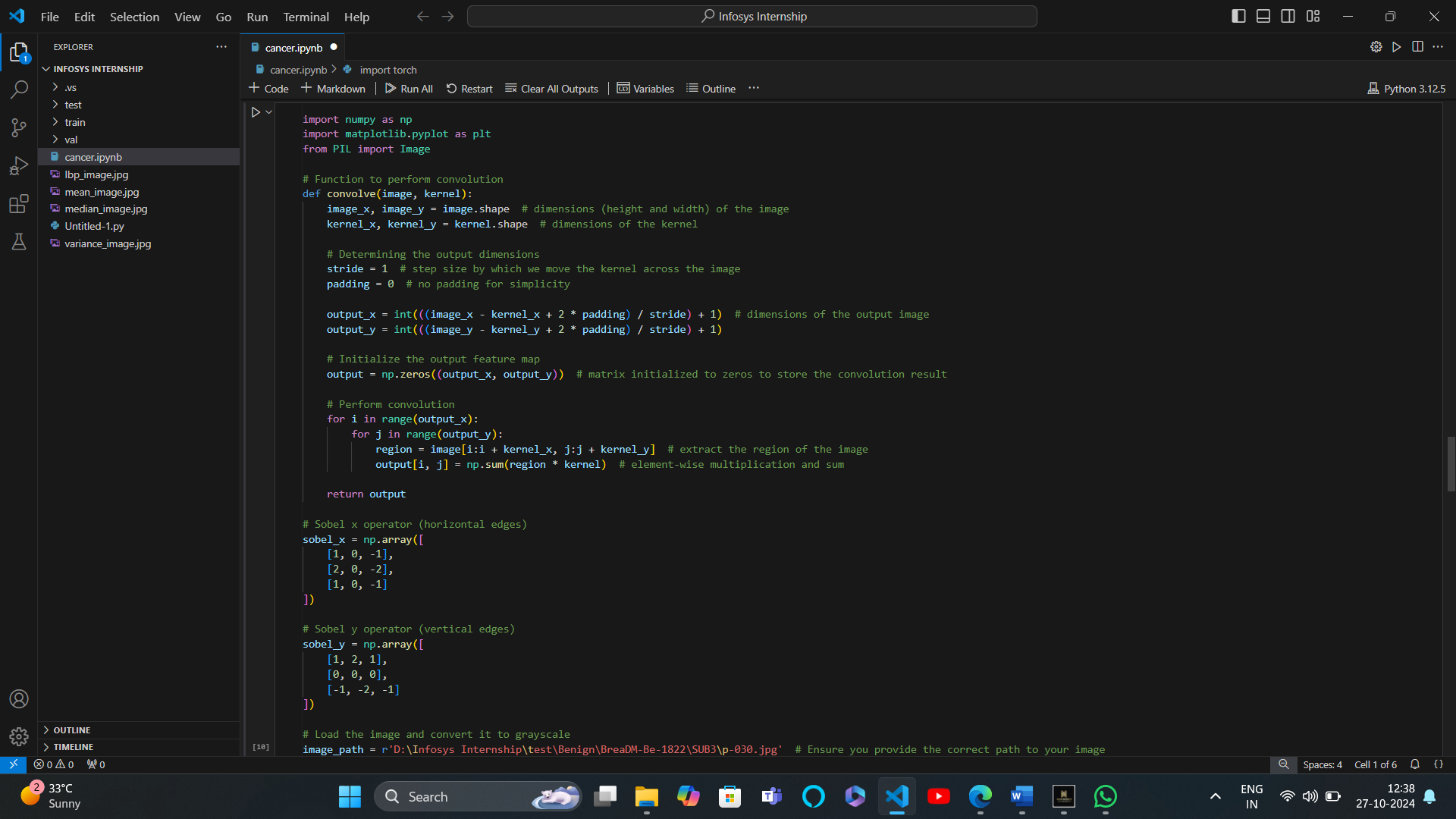


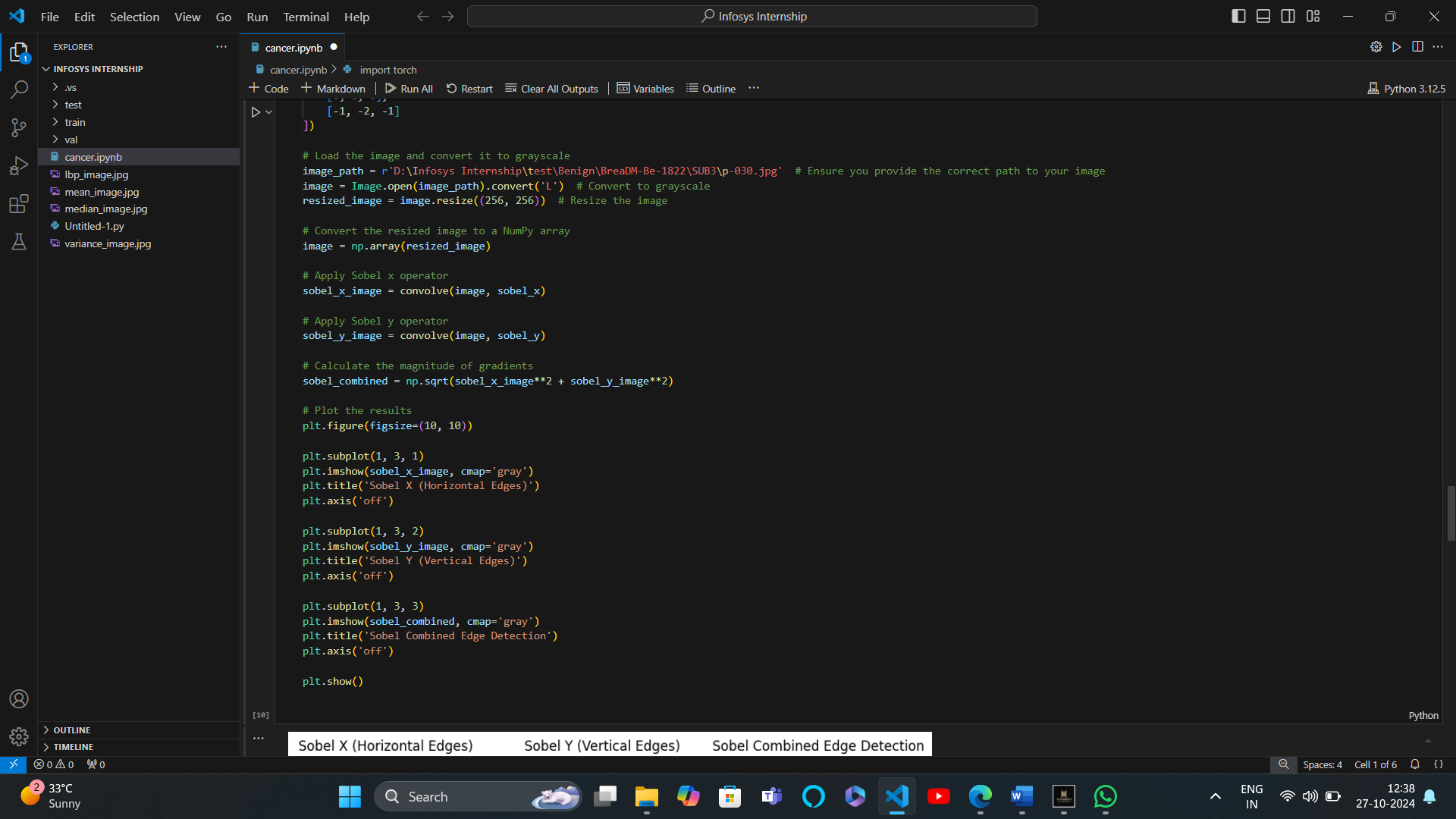


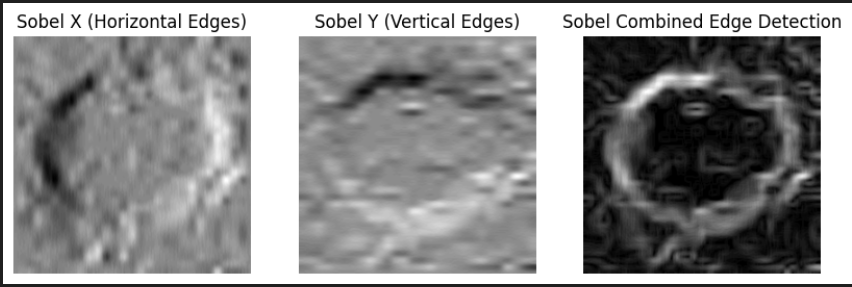
**Day-8**18-10-2024

**Day-9**21-10-2024

Implement Sobel Edge Detection

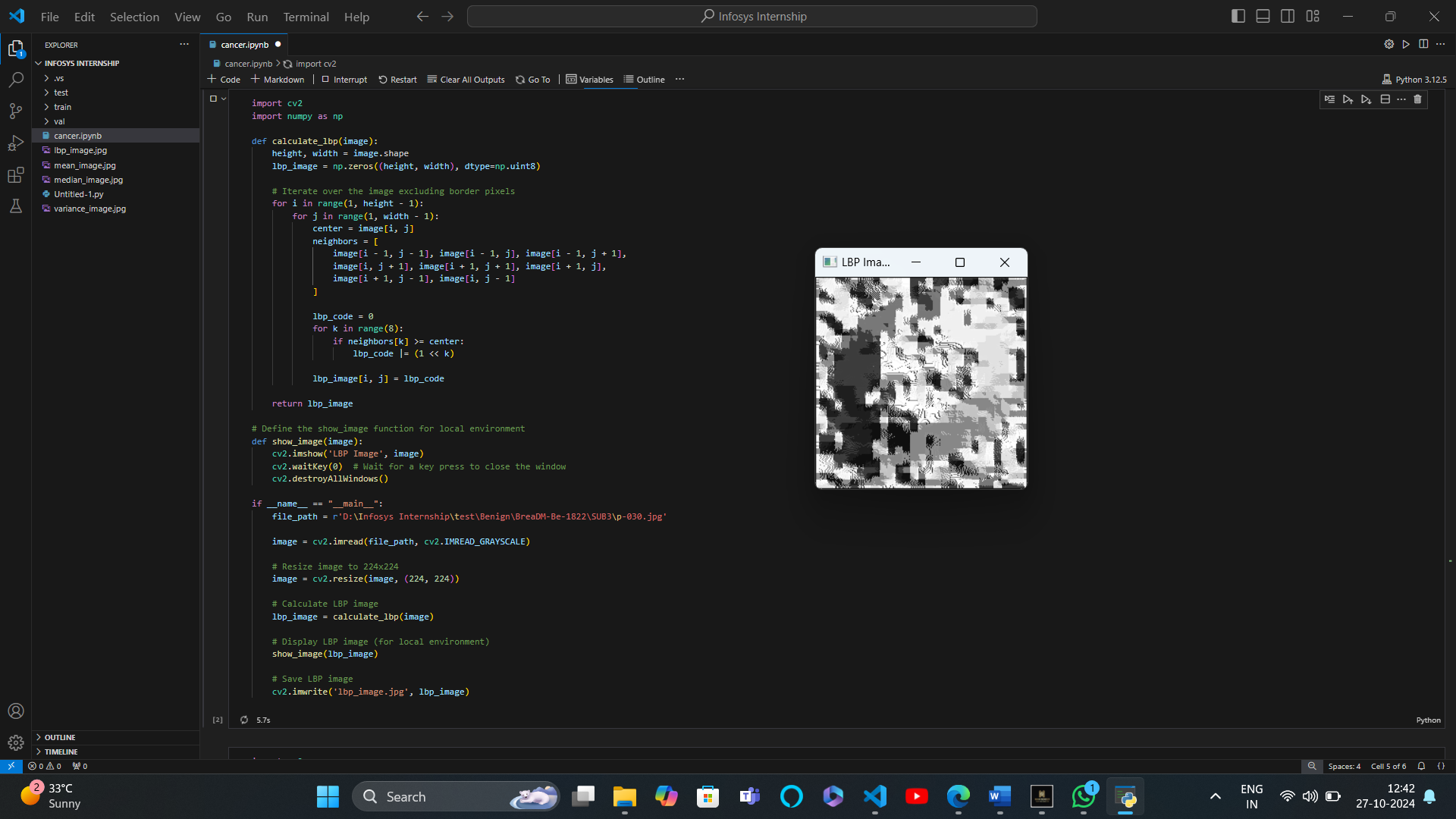






**Day-10**22-10-2024

Implement local binary pattern



Implement mean binary points, median binary points,Variance binary points

