Course:	**Maschinelles Lernen und künstliche Intelligenz/ Machine Learning and Artificial Intelligence**		
(Group) Member:	*Nr.*	*Name*	*Matrikel*
	1	**Miguel Angel Lopez Mejia**	**5197617**
	2	**R.A.G.L.Shehan Ranawaka**	**5195770**

Task Reporting Template

Please read the given Task carefully. You can find all necessary information in Moodle.

1. Definition

1.1. Introduction and Explanations

- Task 6.1 Adding Kitchenware images to the provided database
- Task 6.2 Exploring and applying Image processing methods for feature extraction
- Task 6.3 Preparing the feature extracted images for an ANN

1.2. Allowed Tools

- The script is going to be executed using ***Python version 3.13.2***
- ***Visual Studio Code*** is going to be used as IDE (Integrated Development Environment)
- **Libraries:** pandas, numpy, logging, tensorflow, matplotlib

2. Your Preparation/ Concept

2.1. Structure your Problem

Libraries:

- **OpenCV** (cv2) is a powerful computer vision library that provides tools for image and video processing, including functions for image manipulation, feature detection, object recognition, and machine learning integration. It offers efficient implementations of various image processing algorithms like edge detection, blurring, and thresholding, making it essential for real-time computer vision applications.
- **NumPy** is a Python library providing a multidimensional array object, derived objects like masked arrays and matrices, and tools for fast array operations, including math, logic, shape manipulation, sorting, I/O, Fourier transforms, linear algebra, statistics, and random simulation. In our implementation, it's used for efficient array operations and mathematical computations on image data.
- **Matplotlib** (pyplot) is a comprehensive plotting library for creating static, animated, and interactive visualizations in Python. It provides a MATLAB-like plotting interface and is used in our code to display and compare different image processing results through various plot types and layouts.
- **TensorFlow** is an open-source machine learning framework that provides tools for building and training neural networks. While not directly used in our current implementation, it's important to mention as it's commonly used in conjunction with image processing for deep learning applications, such as image classification, object detection, and image segmentation. TensorFlow can process the preprocessed images from OpenCV to perform advanced AI **tasks.**

2.2. Your Implementation

Virtual Environment

- In order to run the script, a virtual environment must be created, open a new terminal and create it as it is shown:

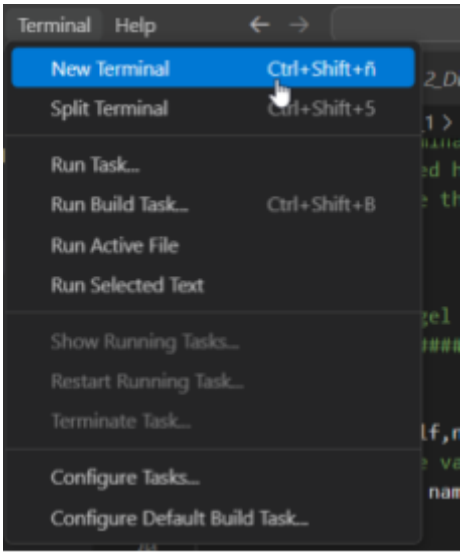

Figure 1: New terminal

- Execute the following command in the console in order to create a new virtual environment: ***python -m venv venv*** once the virtual environment is created, it will be displayed in the explorer section:

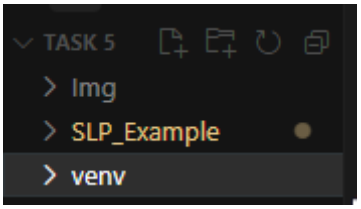

Figure 2: Virtual environment

- In order to activate the virtual environment, execute this script in the console: ***venv/scripts/activate*** .

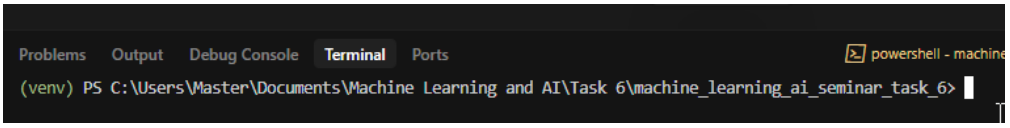

Figure 3: activation of venv

Task 6.1 Adding Kitchenware images to the provided database

Expected Outcomes of Task 6.1

1. The provided baseline of the image database needs to be extended by the team with more images of cups, plates, and dishes [For train and test/validation images]
2. Therefore, for a team of 2, with 3 categories (cups, dishes, and plates), and 4 images for each category needs to be captured and added to the database as a minimum [2 x 3 x 4 = 24 Images (Minimum)]
3. Please use screenshots and document your process thoroughly.

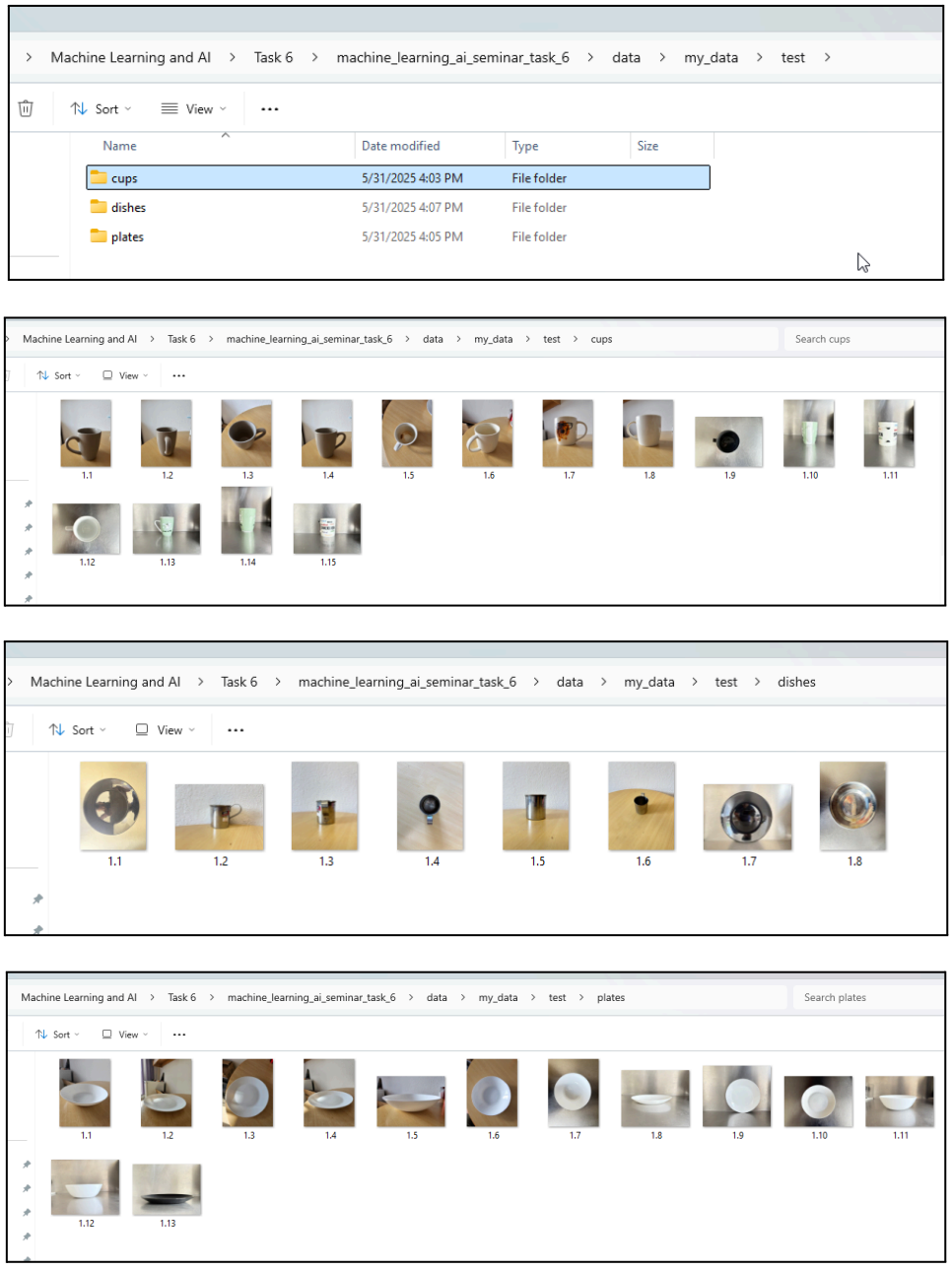

Figure 4: Screenshots of images to use

Task 6.2 Exploring and applying Image processing methods for feature extraction

1. Load your arranged images into your “Task6_2_Kitcheware_Image_processing.py” python code and apply various feature detection such as edge detection and thresholding methods to produce the required features.

```
# Define the categories and paths
categories = ['cups', 'dishes', 'plates']
base_input_dir = 'data/my_data/test'
base_output_dir = 'data/my_data/edge_detected/test'

# Process images for each category
for category in categories:
    # Set up input and output paths for this category
    input_dir = os.path.join(base_input_dir, category)
    output_dir = os.path.join(base_output_dir, category)

    # Create output directory if it doesn't exist
    os.makedirs(output_dir, exist_ok=True)

    # Get all jpg/JPG files from the directory
    image_files = [f for f in os.listdir(input_dir) if f.lower().endswith(('.jpg', '.jpeg'))]

    print(f"\nProcessing {category} images...")
    for image_file in image_files:
        # Construct full file paths
        input_path = os.path.join(input_dir, image_file)
        output_path = os.path.join(output_dir, image_file)

        # Read and process image
        img = cv2.imread(input_path, 0)
        if img is not None:
            print(f"Processing {category}/{image_file}, shape: {img.shape}")
            edges = cv2.Canny(img, 100, 200)
            cv2.imwrite(output_path, edges)
        else:
            print(f"Failed to read image: {category}/{image_file}")
```

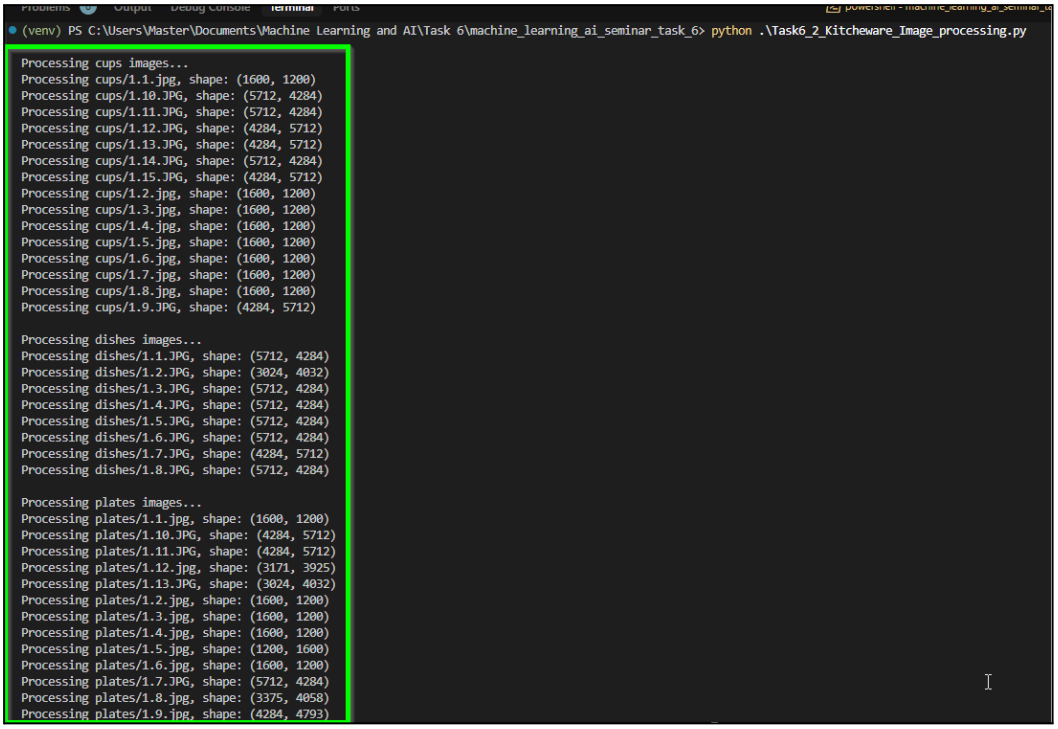

Figure 5: Logs of the images processed

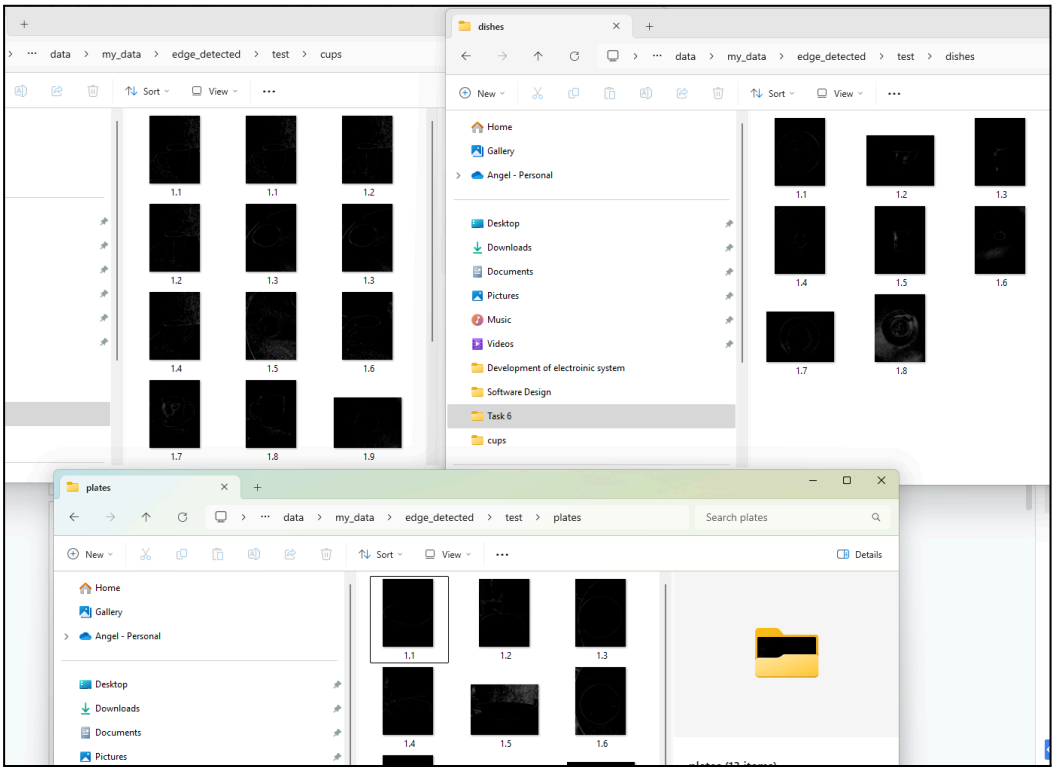

Figure 6: Screenshots of the 3 distinct datasets already processed

Figure 7: Screenshots of the 3 distinct types
of images processed

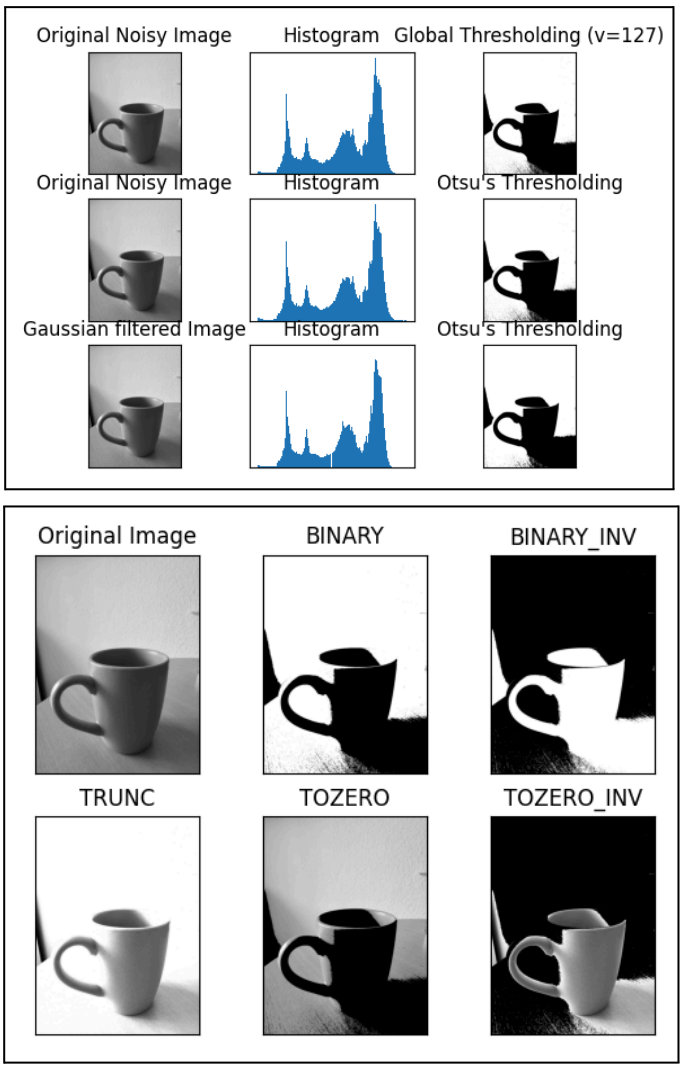

Figure 8: Result of the images processed using the techniques from the python script

2. Other image processing techniques can also be used to acquire the results needed, and it is important for you as a team to explore them. For example, changing the color, geometric transformations, smoothing, filtering and many more.
3. Please determine if there are better ways to extract features from your custom image dataset edge detection and thresholding? Show this by comparing the results from the different image processing techniques.

Expected Outcomes of Task 6.2

1. The code provided only explored edge detection and threshing techniques using OpenCV, more image processing techniques for feature extraction needs to be explored. (Hint:[Changing colour space, geometric transformations, smoothing…etc.])

2. Visualization before and after your image processing, detailed in your documentation. [Have you found better feature extractions?]

Results of analysis

We analyzed 3 distinct techniques for image processing, each one with different variations, all of them described below.

Edge detection

Canny Function

One of the features of the Canny function is that allows to identify edges in an image, but in order to do this, the image needs to be converted to gray scale, then, we used the common threshold ratios 1:2 or 1:3, in this example (50,150), (100,200), (150,250). The first value is called lower threshold, any pixel with a gradient magnitude below this value is considered “not an edge” and the pixel is discarded. The second value is called upper threshold, any pixel with a gradient magnitude above this value is considered “definitely an edge”.

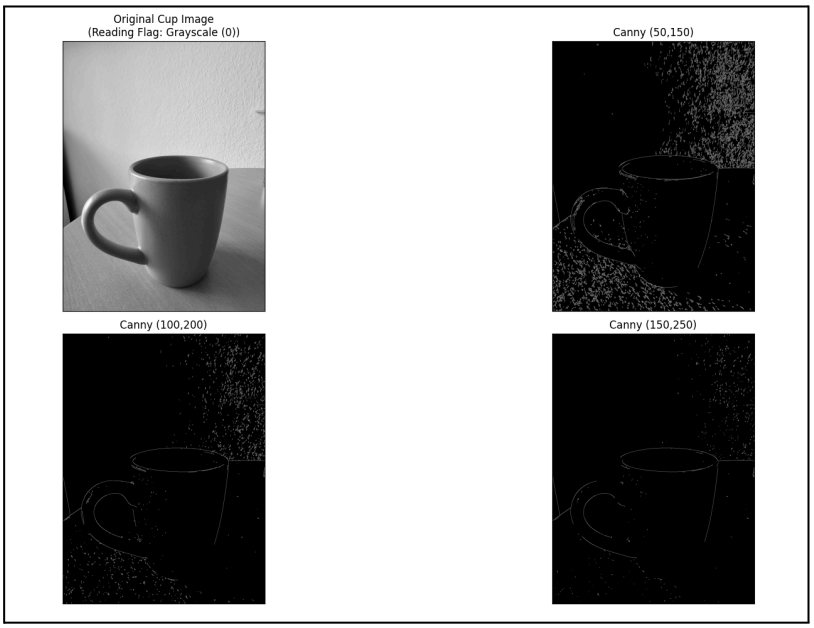

Figure 9: Comparing Canny with different thresholds

From Figure 9, we can appreciate that with higher thresholds, less edges will be detected

Sobel

The Sobel operator is an edge detection algorithm that works by calculating the gradient (rate of change) of image intensity at each pixel. It's particularly good at detecting edges in both horizontal and vertical directions. Detects edges by finding areas where the image intensity changes rapidly. Works by computing the gradient magnitude in both x and y directions

How it works:

- Uses two 3x3 kernels to convolve with the image
- X-direction kernel detects horizontal edges
- Y-direction kernel detects vertical edges
- The stronger the gradient, the stronger the edge

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

X-Direction Kernel

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Y-Direction Kernel

Figure 10: Kernel Schemas

Output:

- Returns gradient values that can be positive or negative
- Higher values indicate stronger edges

In this example it can be seen how the X-direction and Y-direction kernels are generated and the final result:

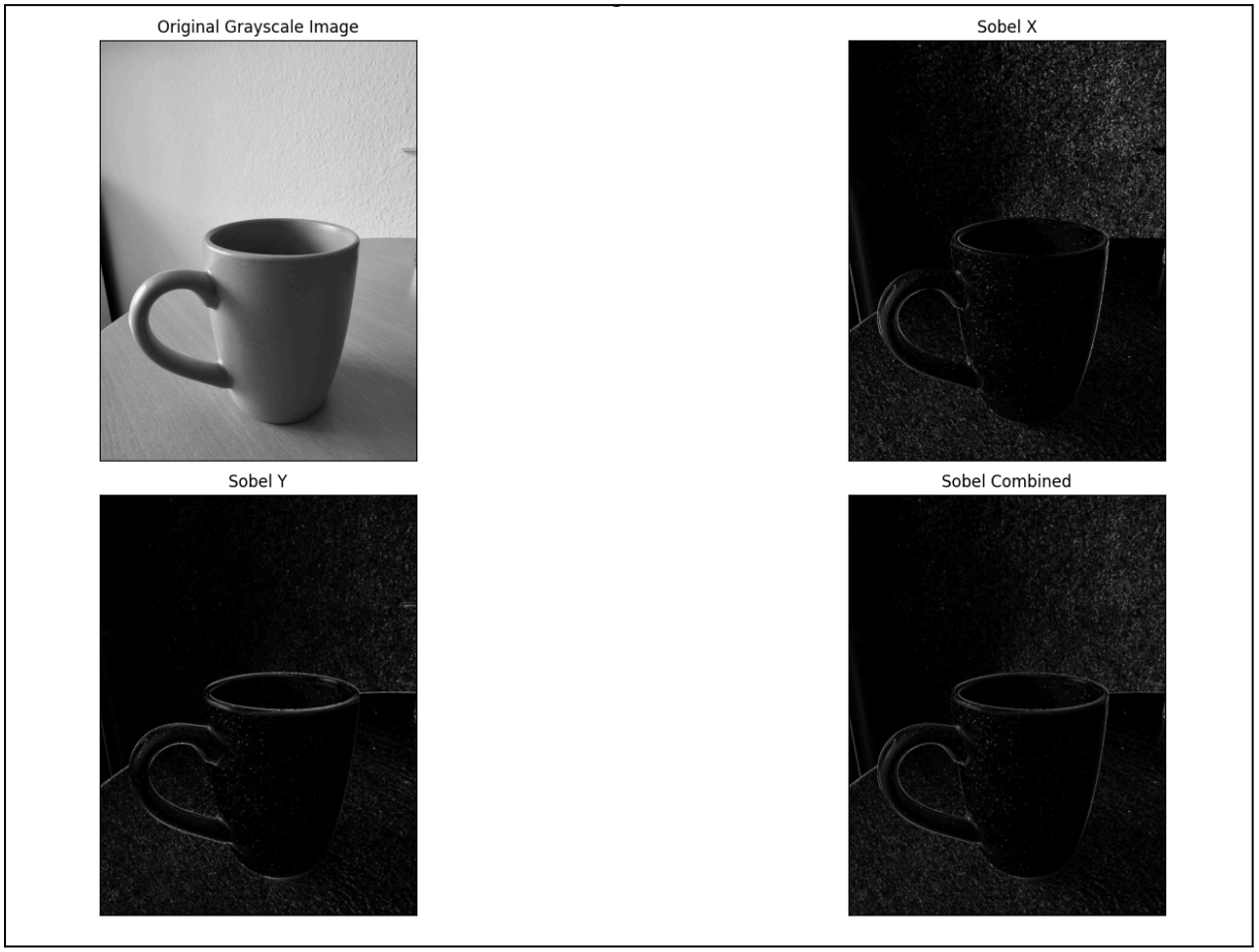

Figure 11: Sobel technique

In this image we can visualize that with different weights, the image shows a better edge detection, in this case when we emphasize the weight in the vertical side.

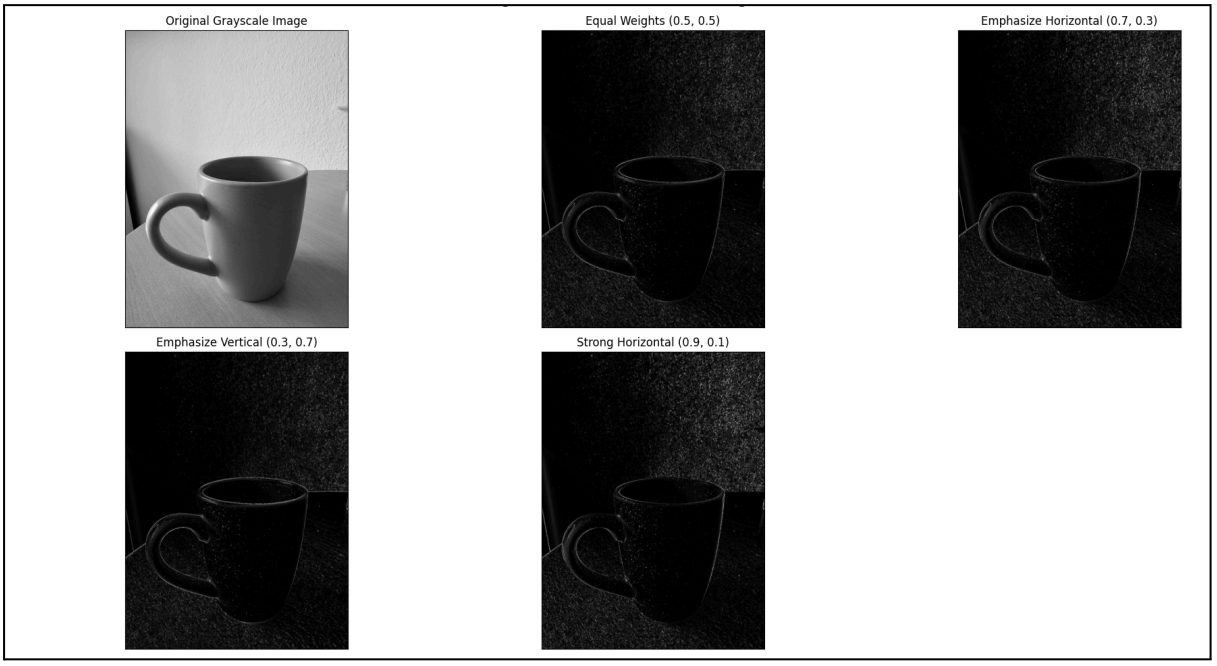

Figure 12: Sobel technique with distinct weights

The code that generates this change in the weight is:

```
dst  = cv2.addWeighted(
       results['Sobel X'], 0.5,
       results['Sobel Y'], 0.5, 0
   )
```

```
results['Sobel X']: First input image (in our case, Sobel X)
alpha: Weight of the first image (0.5 in our code)
results['Sobel Y']: Second input image (in our case, Sobel Y)
beta: Weight of the second image (0.5 in our code)
gamma: Scalar added to the weighted sum (0 in our code)
This is what the code is doing
                dst = alpha * src1 + beta * src2 + gamma

Each pixel in the result is: 0.5 * SobelX + 0.5 * SobelY + 0
The weights (0.5 each) sum to 1.0, which is common for image blending
The gamma value of 0 means no additional brightness adjustment
```

Laplacian

The Laplacian operator is a second-order derivative operator, it measures the rate of change of the gradient (rate of change of intensity), it detects edges by looking for zero crossings in the second derivative of the image.

Mathematical Concept:

For a 2D image, the Laplacian is defined as:

$$\nabla^2 f = \partial^2 f/\partial x^2 + \partial^2 f/\partial y^2$$

Where:

- $\partial^2 f/\partial x^2$ *is the second derivative in* x *direction*
- $\partial^2 f/\partial y^2$ *is the second derivative in* y *direction*

This means it looks at how the intensity changes in both x and y directions simultaneously.

The function in code uses a kernel (matrix) to approximate the second derivatives, for a k=1 the kernel looks like:

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 13: Kernel for laplacian

This kernel is convolved with the image to detect edges.

Edge Detection Process:

The Laplacian highlights regions of rapid intensity change

When the Laplacian output is:

- Zero: No edge (constant intensity)
- Positive: Dark to bright transition
- Negative: Bright to dark transition

We use *np.absolute()* to convert all values to positive for visualization

Kernel Size Effects:

- ksize=1: Uses a smaller kernel, more sensitive to noise

```
[ 0  1  0]
[ 1 -4  1]
[ 0  1  0]
```

- ksize=3: Standard kernel, balanced sensitivity

```
[ 0  0  1  0  0]
[ 0  1  2  1  0]
[ 1  2 -16 2  1]
[ 0  1  2  1  0]
[ 0  0  1  0  0]
```

- ksize=5: Larger kernel, more smoothing, less sensitive to noise

```
[ 0  0  0  1  0  0  0]
[ 0  0  1  2  1  0  0]
[ 0  1  2  3  2  1  0]
[ 1  2  3 -32 3  2  1]
[ 0  1  2  3  2  1  0]
[ 0  0  1  2  1  0  0]
[ 0  0  0  1  0  0  0]
```

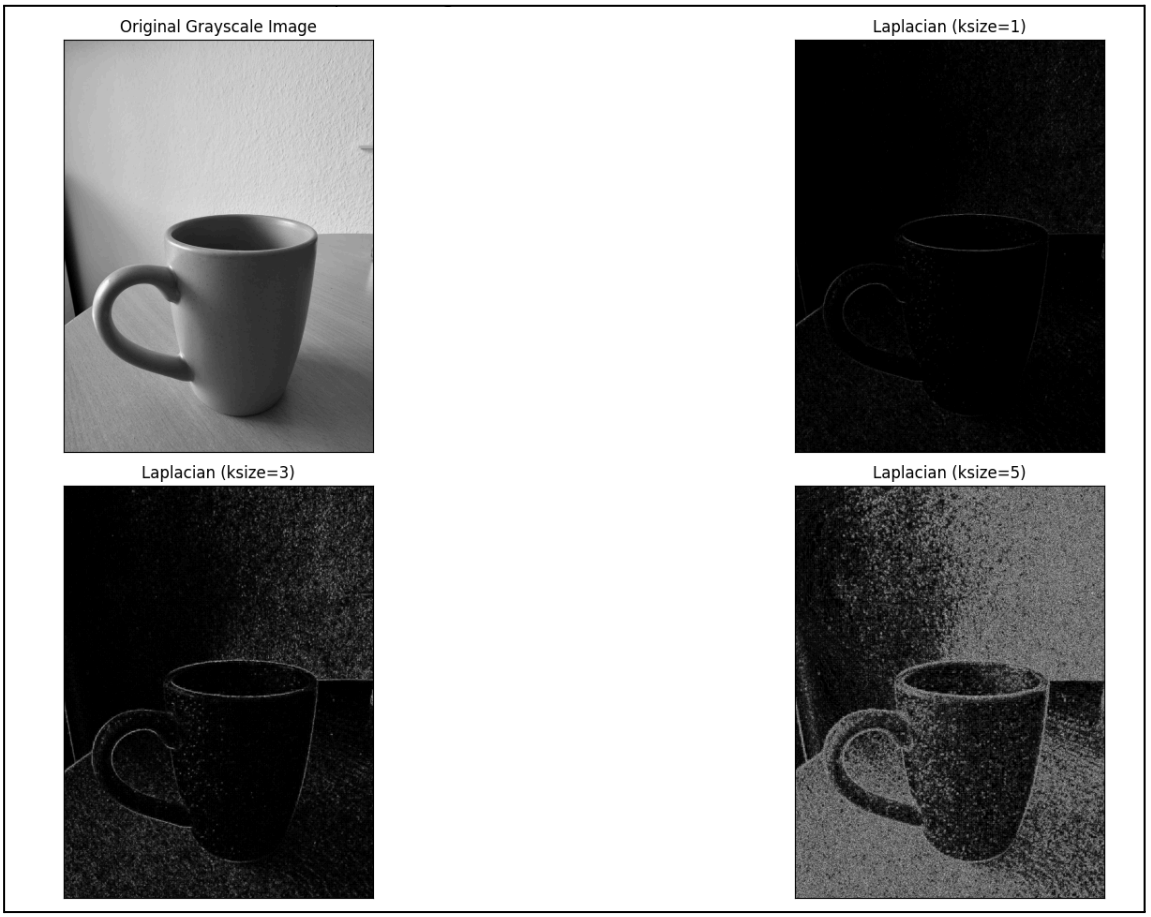

Figure 14: Results of Laplacian using distinct k matrixes

Using this image, we can appreciate that a matrix of 3 offers better results than 1 or 5.

Thresholding

Thresholding is a fundamental image processing technique that converts a grayscale image into a binary image It works by comparing each pixel's intensity value against a threshold value. The main purpose is to separate objects from the background. It's often used as a preprocessing step for other image processing tasks like edge detection, object recognition, and segmentation

Simple/Global Thresholding (Binary)

Is the most basic form of thresholding, uses a single threshold value for the entire image, every pixel is compared against this fixed value

How it works:

- If pixel intensity > threshold: Set to white (255)
- If pixel intensity ≤ threshold: Set to black (0)

Use cases:

- Images with good contrast
- Well-lit scenes
- Simple object detection
- Document processing (e.g., text extraction)

Otsu's Thresholding

Is an automatic thresholding method, determines the optimal threshold value by analyzing the image histogram, based on statistical analysis of pixel intensities.

How it works:

- Calculates the optimal threshold that minimizes intra-class variance
- Assumes the image has a bimodal histogram (two distinct peaks)
- Automatically finds the threshold that best separates the two classes

Use cases:

- Images with varying lighting conditions
- Complex scenes with multiple objects
- When manual threshold selection is not practical
- Medical image processing
- Industrial inspection systems

Figure 15: Results of Binary and Otsu's Thresholding

Smoothing/Blurring

This technique is used to reduce noise and smooth images while preserving important features
The common applications are:

- Noise reduction
- Image preprocessing
- Edge detection preparation
- Feature extraction

Gaussian Blur

Is a Linear filter using a Gaussian kernel, applies weighted averaging based on Gaussian distribution
How it works:

- Each pixel is replaced by a weighted average of its neighbors
- Weights follow a bell-shaped (Gaussian) curve
- Center pixels have higher weights than distant pixels

Use cases:

- General image smoothing
- Noise reduction
- Preprocessing for edge detection

Median Blur

Is a Non-linear filter using median value, replaces each pixel with the median of its neighborhood
How it works:

- Takes a window of pixels
- Sorts their values
- Uses the middle value as the new pixel value

Use cases:

- Salt-and-pepper noise removal
- Medical imaging
- When edge preservation is important

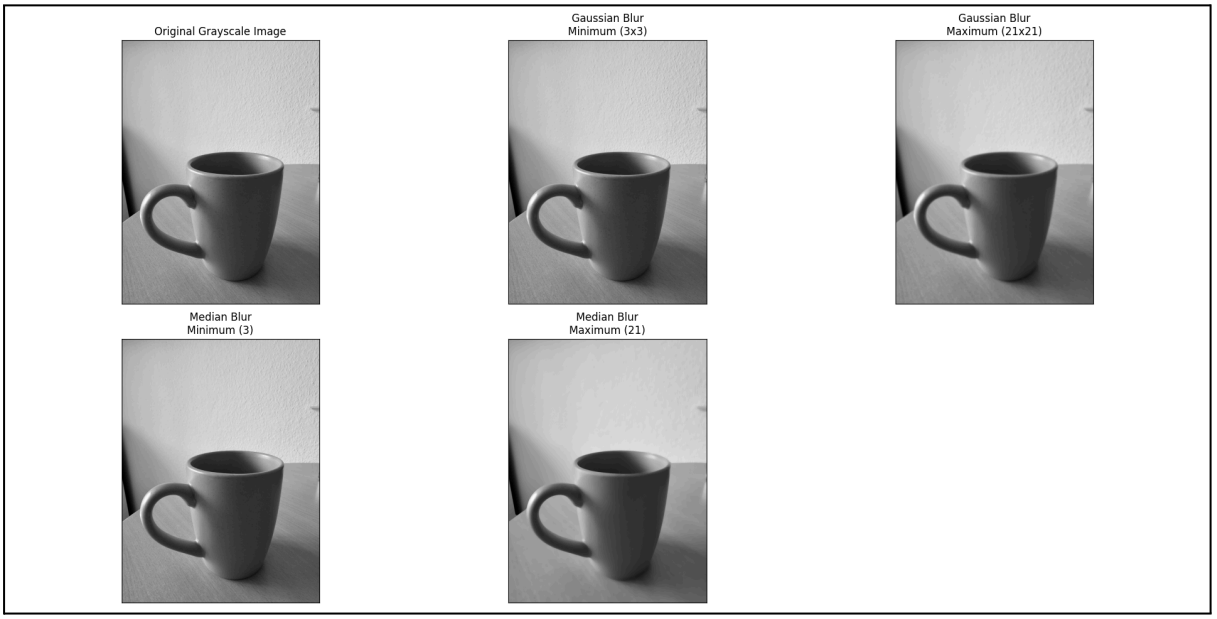

Figure 16: Results of Median Blur vs Gaussian Blur

Task 6.3 Preparing the feature extracted images for an ANN

This Python script is designed to load image datasets from local directories, prepare them using TensorFlow, and visualize sample images from both the training and validation datasets. It uses the **tf.keras.preprocessing.image_dataset_from_directory** utility for efficient dataset preparation and matplotlib for visualization.

Code Explanation

1. Importing Required Libraries

The script imports essential libraries:

- tensorflow: for image preprocessing and dataset handling.
- matplotlib.pyplot: for image visualization.
- os and sys: for file path operations and error handling.

2. Function: set_data_path()

This function defines the expected input image resolution and determines the file paths to the training and test datasets relative to the script's location. It checks whether the training data directory exists; if not, it prints an error and exits the script.

Outputs:

- train_path: Path to training images.
- test_path: Path to test images (edge-detected).
- img_height, img_width: Target image dimensions.

3. Function: prepare_datasets()

This function uses TensorFlow's image_dataset_from_directory() to load and preprocess images from the specified directories into tf.data.Dataset objects.

Key steps:

- Converts images to RGB and resizes them to (128, 128).
- Applies batching and sets a seed for reproducibility.
- Returns two datasets: ds_train for training and ds_validation for validation or testing.

Advantages:

- Efficient loading and transformation of large image datasets.
- Simplifies the pipeline for feeding data into deep learning models.

4. Function: plot_sample_images()

This function displays sample images from both the training and validation datasets to verify that data loading and labeling were successful.

Process:

- Extracts class_names from the datasets.
- Plots a grid of sample images along with their corresponding class labels.
- Uses matplotlib for visualization.

Note: Only the first batch of images is visualized to maintain performance and clarity.

Sample Output:

Code terminal output:

```
PS D:\Anhalt\Machine Learning\Task 6> & "C:/Users/Shehan Ranawaka/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "d:/Anhalt/Machine
earning/Task 6/machine_learning_ai_seminar_task_6/Task6_3_Load_image_and_process.py"
Found 47 files belonging to 3 classes.
2025-06-01 15:30:42.564349: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU
instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the
appropriate compiler flags.
Found 50 files belonging to 3 classes.
['cups', 'dishes', 'plates']
['cups', 'dishes', 'plates']
PS D:\Anhalt\Machine Learning\Task 6>
```

Figure 17: Terminal Output

Key Observations:

- **Files Found:**
    - Training dataset: 47 images
    - Validation (edge-detected) dataset: 50 images
- **Detected Classes:**
  The following 3 class labels were automatically inferred from folder names:

  ['cups', 'dishes', 'plates']

Visualization Output:

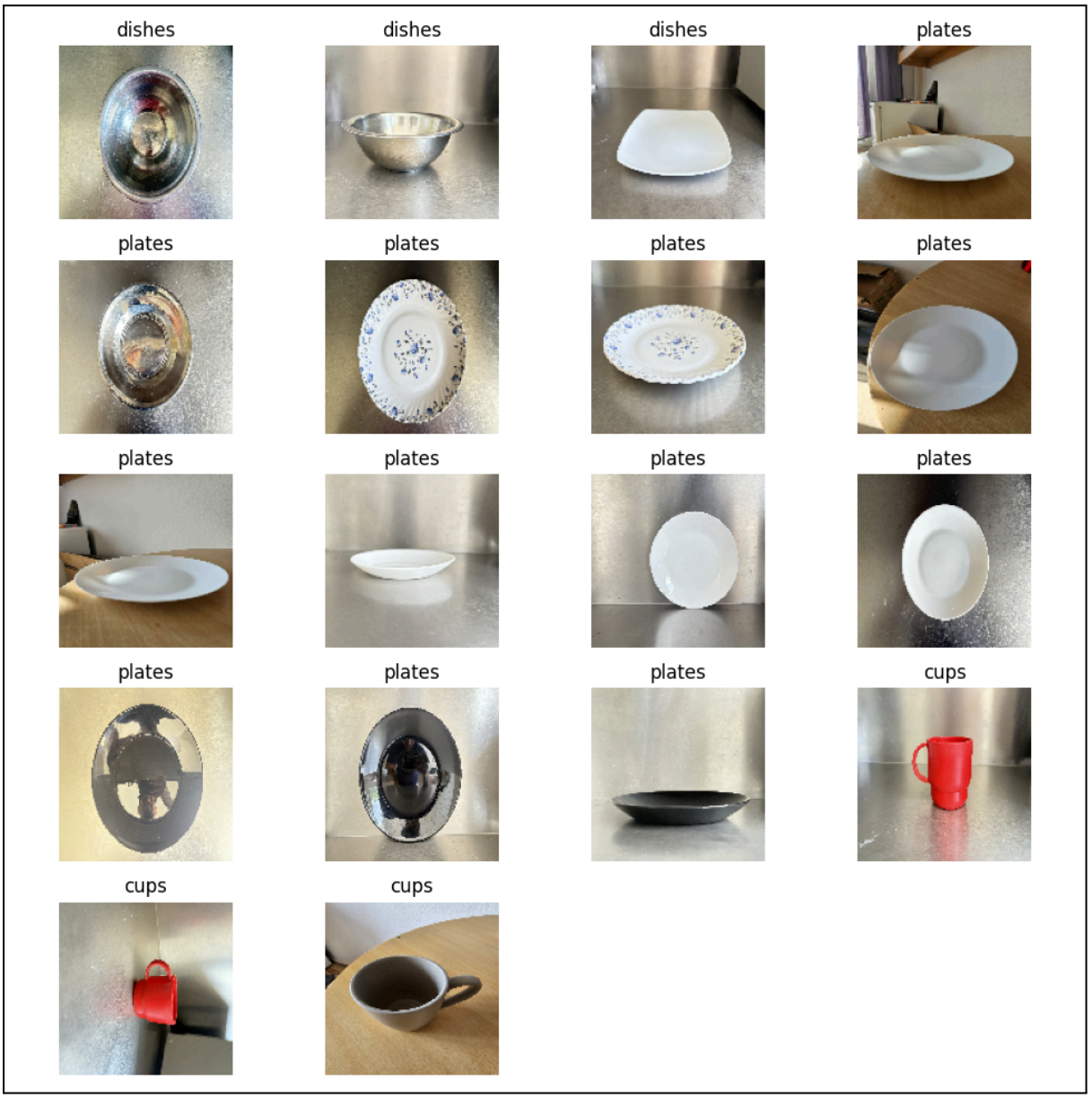

Figure 18: Visualization of the categories used

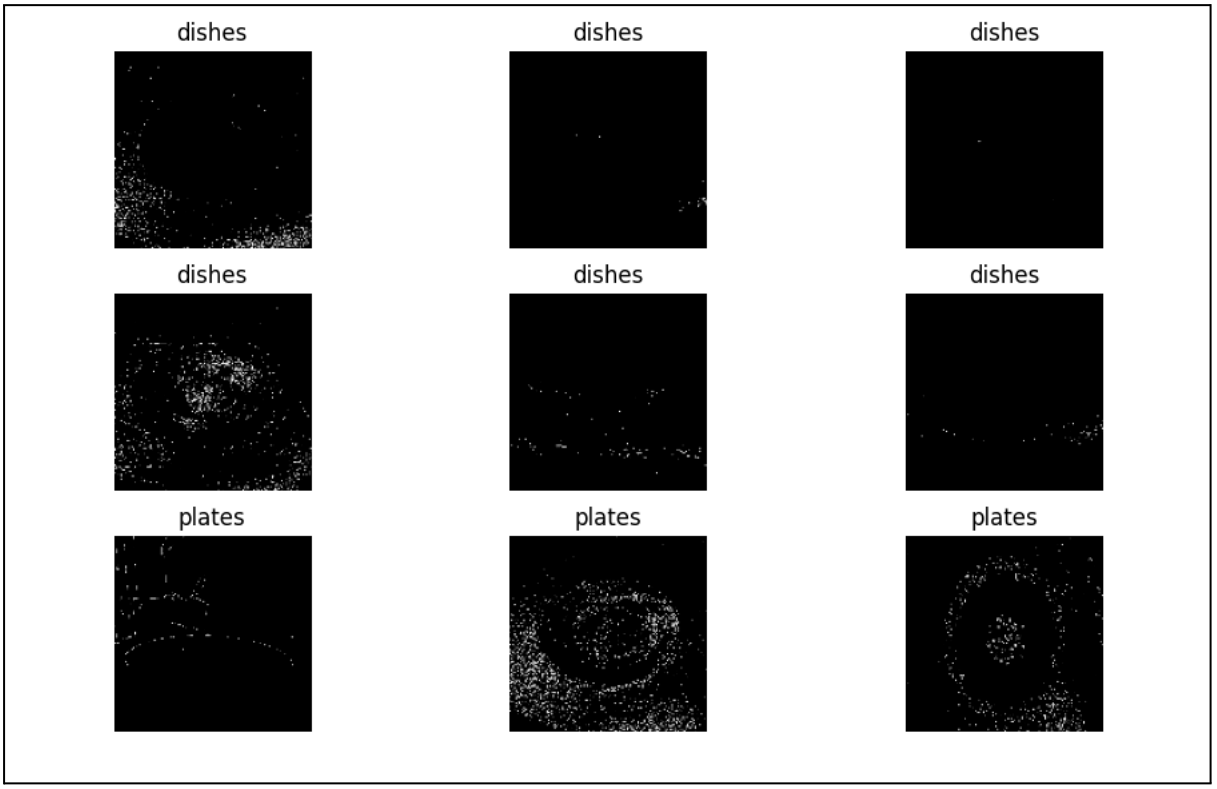

Figure 19: Results after processing

Evaluation/ Testing

In this task, images from three categories—cups, dishes, and plates—were successfully loaded, resized, and visualized using TensorFlow's image processing pipeline. The dataset was split into training and test sets, and class labels were correctly identified and displayed.

Upon visual inspection of the dataset samples, it is evident that there is a class imbalance, with the 'plates' category appearing more frequently than 'cups' and 'dishes'. This imbalance is likely due to an unequal number of images in each class folder. Additionally, since image shuffling was disabled (shuffle=False), the displayed samples do not represent the dataset uniformly and tend to show images from dominant classes first.

This imbalance can negatively impact machine learning model performance, as models may become biased towards more frequently occurring classes. Therefore, balancing the dataset — either by resampling, augmenting underrepresented classes, or applying class weighting during training — is recommended to improve classification accuracy and fairness.

Conclusion

Image processing techniques are fundamental to modern computer vision, serving as crucial preprocessing steps that enhance image quality, reduce noise, and extract meaningful features through methods like blurring, edge detection, and thresholding. These techniques find critical applications in medical imaging (tumor detection), industrial quality control, security systems (face recognition), and autonomous vehicles, where they help machines understand and process visual information effectively. The choice of technique significantly impacts analysis outcomes, with each method (Gaussian blur, Median blur, Canny edges, etc.) serving specific purposes like noise reduction, edge preservation, or object segmentation.

Github Repository

MiguelAnhalt/machine_learning_ai_seminar_task_6

References, extra reading links, and resources

- Dua, D., & Graff, C. (2017). *UCI Machine Learning Repository* http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.
- NIST/SEMATECH. (n.d.). *Box Plot*. In *Engineering Statistics Handbook*. Retrieved May 2, 2025, from https://www.itl.nist.gov/div898/handbook/eda/section3/boxplot.htm
- GeeksforGeeks. (2023, November 20). *Iris Dataset*. https://www.geeksforgeeks.org/iris-dataset/
- W3Schools. (n.d.). *Python Classes and Objects*. https://www.w3schools.com/python/python_classes.asp
- GeeksforGeeks. (2024, October 22). *Activation functions*. Retrieved from https://www.geeksforgeeks.org/activation-functions/
- OpenCV. (n.d.). *Image Processing in OpenCV*. OpenCV-Python Tutorials. Retrieved June 2, 2025, from https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html
- OpenCV. (n.d.). *Image Processing in OpenCV*. OpenCV Documentation. Retrieved June 2, 2025, from https://docs.opencv.org/master/d2/d96/tutorial_py_table_of_contents_imgproc.html
- OpenCV. (n.d.). *Canny Edge Detection*. OpenCV Documentation. Retrieved June 2, 2025, from https://docs.opencv.org/master/da/d22/tutorial_py_canny.htm