Assignment 1

Instructions:

1. **Submission Requirements:**

   Upload the Jupyter notebook in HTML or PDF format with outputs and its corresponding PDF.

   You may also choose to use Google Colab for working on the project.

   Data files do not need to be uploaded with the submission.

   Ensure the updated Jupyter notebook is submitted with properly formatted and aligned outputs. Incomplete outputs, misalignments, or poorly written comments will result in a deduction of marks.

   Partial code and partial output will be evaluated, and marks will be awarded based on the PDF file.

   Only the latest submission would be considered for marking.

2. **File Naming Convention:**

   Name the file as: CV_assignment1_group_problemstatmentnumber

3. **Plagiarism Policy:**

   Any form of plagiarism will be taken very seriously, resulting in zero (0) marks.

   All submissions must be the result of your original effort.

   Copying from any sources, whether online or from peers, is strictly prohibited.

   Unauthorized collaboration to gain an unfair advantage is prohibited.

   Both the person sharing resources and the one receiving them will face consequences for plagiarism.

   Identical or significantly similar submissions will be investigated, and severe punishments will be imposed on those found guilty.

4. **Late Submission Policy:**

   No late submission reports will be evaluated.

5. **Queries:**

   For any questions regarding the assignment, use the discussion forum.

6. **Programming Instructions:**

   Use appropriate Python programming techniques to read and process the dataset.

   Convert the notebook into HTML or PDF format along with outputs and upload the file.

Avoid submitting excessively long notebooks by eliminating irrelevant lengthy prints.

7. Individual student contributions should be listed at the end of the report - else will be awarded 0

Goals:

- Experiment with additional features like texture analysis using Gabor filters. (**Optional**)
- Visualize feature importance if using models like Random Forest or SVM. Hint : Show a bar plot of Model Accuracies and metrics for different Feature Engg techniques . Like for instance : HOG,LBP,Edge Det , HOG+ LBP , LBP+Edge det , HOG+LBP+Edge det

Emphasize handcrafted feature engineering and their role in solving computer vision problems. This assignment is not about achieving perfect accuracy but understanding how handcrafted features contribute to scene classification. Let your analysis and insights shine!

Problem Statement Selection:

Choose **ANY ONE** problem statement from the three provided below.

==

Criteria	**Description**	**Max Points**
Import Required Libraries	Import necessary libraries for image processing (e.g., OpenCV, skimage) and machine learning (e.g., scikit-learn, pandas, matplotlib). Assessor Note: Check if all required packages are correctly imported and commented.	0.5
Data Acquisition	Load and structure the dataset. Print dataset size and category-wise image count. Plot distribution of labels (bar/pie chart). Assessor Note: Verify correct handling of directory structure and label mapping.	0.5

Data Preparation	Resize images, convert to grayscale, apply histogram equalization, and other preprocessing techniques. Perform an 80-20 stratified split for training/testing. Assessor Note: Look for efficient pipeline implementation and justification for preprocessing choices.	1.0
Feature Engineering	Extract handcrafted features: LBP, HOG, Edge detection (Sobel/Canny). Normalize features (Min-Max or Z-score). Optional: PCA or dimensionality reduction. Assessor Note: Check clarity and correctness of feature extraction and structuring.	2.5
Model Building	Use one classical ML algorithm (e.g., SVM, Random Forest, XGBoost). Train the model on the engineered features. Assessor Note: Check for proper training code, use of cross-validation, and comments.	1.5
Validation Metrics	Print Accuracy and F1-score for test data. Assessor Note: Ensure metric usage is appropriate and consistent with predictions.	0.5
Model Inference & Evaluation	Randomly pick 5 test images. Display predicted vs. actual labels using the best model. Justify why certain features performed better. Assessor Note: Look for visual clarity and reasoning.	1.0
Validation of actual test	Input the test image created by you and justify the models performance	1.5
Documentation, Presentation & Code Quality	Well-documented notebook or report. Clear presentation (markdown/text sections, headings, plots). Clean and readable code. Assessor Note: Reward attention to formatting, modularity, and clarity.	1
		Total 10

Problem 1

Handwritten English Character Recognition Using Classical Vision & Machine Learning

Goal

Develop a classical computer-vision-based recognition system for handwritten English letters (A–Z). The system must use low-level and mid-level feature extraction and train classical ML models on the EMNIST Letters dataset. The trained system will then evaluate a real handwritten sample provided by the student.

Required Concepts and Tools
Students must demonstrate understanding and implementation of the following:

Preprocessing & Low-Level Vision:

- Grayscale conversion
- Histogram normalization / equalization
- Gaussian or median smoothing

Mid-Level Feature Extraction:

- Edge detection (Sobel, Canny)
- Contour/connected components
- Texture descriptors (LBP)
- Keypoint descriptors (ORB)

Machine Learning Models:

- k-NN
- SVM
- Random Forest
- Logistic Regression

Assignment Tasks

Part 1 – Dataset Selection & Preparation

1. Load the EMNIST Letters dataset.
2. Optionally sample 10k–20k images.
3. Normalize images to 28×28 grayscale.

4. Ensure labels correspond to characters A–Z.
5. Collect one real handwritten test image (scanned or photographed).

Part 2 – Preprocessing and Feature Extraction

Apply classical image processing operations:

Low-Level Features:

- Convert to grayscale
- Apply histogram normalization/equalization
- Apply Gaussian or median blur

Mid-Level Features:

- Sobel/Canny edge detection
- Extract contours or connected components
- Compute texture descriptors like LBP
- Compute ORB keypoints if applicable

Goal: Create a feature vector of 3–5 features per image and store all features in a matrix.

Part 3 – Model Training

Train the following classical ML classifiers using extracted features:

- k-NN
- SVM
- Random Forest
- Logistic Regression

Use training/validation split or k-fold CV.

Part 4 – Model Evaluation

Compute:

- Accuracy
- Precision, Recall, F1-score
- Confusion Matrix
- Performance on the handwritten test letter generated by you

---

Part 5 – Analysis & Discussion

Discuss:

- Most effective feature combinations
- Segmentation or noise challenges
- Accuracy difference: dataset vs. real sample
- Suggested improvements (e.g., better descriptors, hybrid deep learning approaches)

Problem 2

The goal of this assignment is to develop a robust system for detecting straight lane lines in road images using purely classical computer vision and Machine Learning (ML) techniques. The implementation must rely on concepts of **Edge Detection, **Hough Transformation**, and **Data Clustering/Averaging** for line fitting.**

Dataset -
(https://www.kaggle.com/datasets/dataclusterlabs/lane-detection-road-line-detection-image-dataset?resource=download)

Required Concepts and Tools
Students must demonstrate understanding and implementation of the following:

1. Preprocessing: Color space conversion (e.g., to HLS or HSV) and Gaussian Blurring.
2. Edge Detection: Implementing or utilizing the Canny Edge Detector.
3. Feature Extraction: Defining and applying a Region of Interest (ROI) mask.
4. Line Parameterization: Implementing or utilizing the Standard or Probabilistic Hough Transform (HT).
5. Machine Learning Filtering: Using statistical methods or simple ML techniques (e.g., slope-based classification and **Averaging/RANSAC/K-Means**) to group and fit the lines.

Assignment Tasks
Part 1: Preprocessing and Edge Detection

A. Load and Convert: Load a sample image. Convert it to grayscale and apply a Gaussian Blur to reduce noise. Experiment with converting the image to the HLS or HSV color space to enhance the visibility of white/yellow lines.
B. Canny Edge Detection: Apply the Canny Edge Detector. Clearly state the chosen high and low thresholds τ_{high} and τ_{low}.
C. Region of Interest (ROI): Define a trapezoidal or triangular polygonal mask. Apply this mask to the Canny edge map to focus computation only on the road area in front of the vehicle.

Part 2: Line Detection via Hough Transformation

A. Hough Transform: Apply the Hough Transform to the masked edge image. The output should be a set of line segments defined by parameters like (ρ, θ) or (x1, y1, x2, y2).
B. Parameter Space: Briefly explain how a line in the image space (x, y) maps to a point in the Hough space (ρ, θ) using the equation:

$$\rho = x \cos \theta + y \sin \theta$$

Part 3: Data Clustering and Lane Averaging (ML Focus)

This step treats the detected line segments as data points that require statistical grouping and modeling to identify the dominant lane lines.

A. Slope Calculation and Classification: For each detected line segment, calculate its slope (m). Classify the line segments into two primary groups based on their slope:
   - Left Lane Group: Lines with a negative slope ($m < -\varepsilon$).
   - Right Lane Group: Lines with a positive slope ($m > +\varepsilon$).

   Discard lines with near-zero slopes ($|m| \leq \varepsilon$) as noise or irrelevant horizontal features.
B. ML-Based Line Fitting: Use one of the following methods for each of the two groups (Left and Right) to determine a single, final representative line:
   - Simple Averaging: Calculate the **mean slope** (¯m) and **mean intercept** for all lines within each group.
   - Robust Fitting (Advanced ML): Implement a simplified RANSAC (Random Sample Consensus) algorithm or apply **k-means clustering** (with k = 1 for each group) on the (slope, intercept) feature space to find the single best-fit line parameters (¯m, ¯b) for the left and right lanes, thus making the result robust to outliers.
C. Projection: Use the two final calculated line parameters (¯mleft, ¯bleft and ¯mright, ¯bright) to extrapolate and draw the two lane lines onto the original color image over the defined ROI.

Deliverables

1. Source Code: Fully commented code implementing the entire pipeline.
2. Output Image: The final image for at least three different test cases, clearly showing the detected and overlaid lane lines.
3. Technical Report (PDF): A brief, one-page report detailing:
   a. The values chosen for Canny thresholds and Hough parameters.
   b. • A mathematical justification for the chosen ML-based line fitting method (Part 3b).
   c. • An analysis of the system’s performance, including any observed limitations (e.g., curved roads, shadows).

Satellite Image Classification (Urban vs. Natural) Using Classical Vision + ML

Goal

Build a classical machine-learning system that classifies satellite images into Urban or Natural categories using handcrafted image features. The system must incorporate

pixel-level, texture, and gradient-based features and evaluate performance on a real external image.

Dataset - Eurosat data (https://www.kaggle.com/datasets/apollo2506/eurosat-dataset)

Required Concepts and Tools

Preprocessing:

- Resize to 128×128
- Grayscale conversion
- Histogram Equalization or CLAHE
- Gaussian/Median smoothing

Feature Extraction:

Low-Level:

- Intensity histogram
- LBP texture patterns
- Edge detection (Sobel/Canny)

Mid-Level:

- HOG descriptors
- Optional: SIFT/ORB features

Feature Selection & Dimensionality Reduction:

- PCA or variance thresholding

Machine Learning Models:

- SVM
- Random Forest
- 5-fold cross-validation

Assignment Tasks

Part 1 – Dataset Preparation

1. Use the fMoW dataset.
2. Select ~500 images each for Urban and Natural.

3. Create an 80:20 train-test split.
4. Keep one external real-world image for final evaluation.

Part 2 – Preprocessing

Apply classical preprocessing:

- Resize each image to 128×128
  Convert to grayscale
- Perform histogram equalization / CLAHE
- Apply Gaussian or Median filtering

Part 3 – Feature Engineering

Compute the following features:

Low-Level Features:

- Pixel intensity histogram
  LBP texture descriptor
- Edges using Sobel/Canny

Mid-Level Features:

- Histogram of Oriented Gradients (HOG)
- Optional SIFT/ORB descriptors

Store extracted features in CSV/NumPy/Pickle format.

Part 4 – Feature Selection

Use:

- PCA to reduce dimensionality, or
- Variance thresholding to eliminate low-information features

---

Part 5 – Model Training

Train two models using 5-fold cross-validation:

- Support Vector Machine (SVM)
- Random Forest

Part 6 – Model Evaluation

Compute:

- Accuracy
- Precision
- Recall
- F1-score

- Confusion matrix
- Compare model predictions on the external real-world image you clicked
- Visualize results (bar charts, confusion matrix heatmap).

Part 7 – Analysis & Discussion

Discuss:

- Most discriminative feature combinations (LBP + HOG usually best)
- SVM performance trends
- Misclassifications and limitations
- Recommendations for future hybrid models (handcrafted + CNNs)