Dirty Dish Neural Network Classifier

Cristian Madrazo

Final Project, CSCI 575: Adv. Machine Learning
Prof. Zibo Wang

December 6, 2024

Abstract

Dirty Dish Neural Network Classifier
This project explores the application of machine learning in automating the binary classification of dish cleanliness. Using a Convolutional Neural Network (CNN) architecture, the model is trained on minimally processed and SIFT-enhanced datasets. The primary goal is to develop a web application that allows users to upload dish images and receive real-time cleanliness classification. This approach improves efficiency, reduces resource waste, and ensures consistent hygiene standards. This project also evaluates the performance impact of integrating SIFT for feature extraction.

Contents

Introduction

Problem Statement

Problem statement: Can I use scale-invariant feature transform (SIFT) to enhance the performance of a convolutional neural-network algorithm in performing the binary classification of images of dishes as clean or dirty?

Non-technical Problem

In many households and commercial kitchens, ensuring that dishes are thoroughly cleaned is crucial for maintaining hygiene and preventing contamination. Manual inspection of dishes to check if they are clean can be time-consuming and inconsistent, especially in high-volume and/or high-throughput settings. Additionally, the rise of awareness of resource consumption raises the question of ethics, not to mention efficiency and cost, of washing an already clean dish. An automated system to classify dishes as "dirty" or "clean" would be beneficial to aid in efficiency of hygiene systems and efforts to conserve resources.

Technical Problem

There exists an incompatibility of the SIFT feature output with the input requirements of CNNs. Specifically, SIFT generates keypoints and descriptor vectors with variable lengths, as the number of keypoints varies across images. CNNs, however, require fixed-size, homogeneous input tensors. This mismatch makes it challenging to directly use SIFT features in a CNN.

Scope

The scope of this project involves designing, implementing, training and testing a model. Given the class coursework and topics, a CNN (Convolutional Neural Network) was chosen, arbitrarily, to be used in this project.

Additionally, a feature extraction techniques were explored.

A big part of the scope of this project is to provide the user with an accessible interface to the model. As proposed, an API (Application Programming Interface) is not sufficient in this project, so a GUI (Graphical User Interface) was developed.

Part of training the model includes gathering and preparing relevant datasets, which is within the scope of this project as well.

Outcomes/Objectives

The objective of this project a web application where a user can interact with a trained model. On this application the user can upload images and have them classified by the neural network.

Additionally, an outcome I hope to achieve is to test the applicability of SIFT (Scale Invariant Feature Transform) as a feature extraction technique for this application. If it is deemed to improve performance, than the neural network in the backend of the application should use this technique.

Methodology

The methodology for this project was as follows:

Data Preparation

1. Find a suitable dataset
2. Modify and clean the data, prepare it for processing
3. Minimally process the data (ensure shape homogeneity)

Feature Extraction Exploration

1. Make a copy of the data and process with SIFT and a tailored feature synthesis technique to return data to homogeneity
2. Build 2 CNN models ready to accept the minimally processed dataset and the SIFT dataset

3. Train and evaluate models, compare against each other, choose best performing model

Development

1. Develop an image classification program off of the best performing model

2. Build a web application program that communicates with the image classification program

Background

CNNs (Convolutional Neural Networks)

A **Convolutional Neural Network (CNN)** is a special type of neural network designed for processing structured data (note, CNN architecture requires homogeneity in the input data), such as images. CNNs excel in tasks like image classification, object detection, and video analysis because they are specifically structured to recognize patterns and spatial hierarchies in the input data.[5] This is why they were a perfect candidate for this project, where it was not within the scope to test several types of models.

- **Convolutional Layers**:
  - The primary building blocks of CNNs.
  - These layers apply convolutional filters (kernels) to extract features like edges, textures, or patterns from the input.[4]
  - Filters slide across the input image, capturing spatial and local dependencies.
- **Pooling Layers**:
  - Also known as subsampling or downsampling layers.[4]
  - These reduce the spatial dimensions of the feature maps, making the network computationally efficient and robust to small translations in the input.[5]
  - Common types:
    - **Max Pooling**: Takes the maximum value in a region.[4]
    - **Average Pooling**: Takes the average value in a region.[4]
- **Activation Functions**:
  - Non-linear functions applied after convolutions to introduce non-linearity.

  - Common functions: **ReLU (Rectified Linear Unit)**, Sigmoid, and Tanh. [5]

- **Fully Connected Layers**:
  - Dense layers where every neuron is connected to every neuron in the previous layer.[5]
  - Used at the end of the network to classify the extracted features into specific categories.
  - The corresponding activation function should match the objective, in this project I aim to perform **binary classification**

- **Dropout (Regularization)**:
  - A technique to prevent overfitting by randomly deactivating some neurons during training.[5]

SIFT (Scale-Invariant Feature Transform)

Scale-Invariant Feature Transform (SIFT) is a computer vision algorithm used to detect and describe local features in images. Developed by David Lowe in 1999, SIFT is widely utilized in tasks such as object recognition, image stitching, and 3D reconstruction due to its robustness to changes in scale, rotation, and illumination.

Key Steps of the SIFT Algorithm

1. **Scale-Space Extrema Detection**:
   - SIFT identifies potential keypoints by detecting extrema in a scale-space representation of the image.[1]
   - The scale-space is generated by applying a Gaussian filter at different scales to the image.[1]
   - Keypoints are located where the Difference of Gaussians (DoG) reaches a local maximum or minimum across both spatial and scale dimensions.[1]
2. **Keypoint Localization**:
   - Potential keypoints are refined to improve stability.

- Low contrast points and edge-like structures are removed to ensure robust feature detection.[1]
- This step uses the Taylor series expansion of the DoG to locate the keypoints with sub-pixel accuracy.[1]

3. **Orientation Assignment**:
   - Each keypoint is assigned one or more orientations based on the gradient directions of the surrounding pixels.[1]
   - This ensures that the descriptors are invariant to image rotation.[1]
4. **Keypoint Descriptor Generation**:
   - A feature vector is created for each keypoint by computing the gradients in a local neighborhood.
   - The gradients are divided into orientation histograms, creating a descriptor that is robust to minor deformations and lighting changes.[1]
   - The OpenCV implementation (eg. the algorithm implementation used in this project) defaults to a descriptor size of 128
5. **Matching Keypoints**:
   - The final descriptors are compared across images using a distance metric, such as the Euclidean distance.[1]
   - Matching pairs of keypoints are identified based on the nearest neighbor ratio test to ensure reliable correspondences.[1]

Advantages of SIFT

- **Scale and Rotation Invariance**: SIFT features remain stable across a wide range of scales and rotations.[1]
- **Robustness to Illumination Changes**: The algorithm is designed to handle changes in lighting and minor affine transformations.[1]
- **Wide Applications**: SIFT is applicable in tasks like object detection, image matching, and panoramic image stitching.

Limitations of SIFT

- **Computational Complexity**: SIFT is relatively slow compared to modern feature extraction techniques, making it less suitable for real-time applications.
- **Patent Restrictions**: The algorithm was patented until 2020, limiting its use in certain applications during that time.[1] Of course, this project takes place in 2024 so I are not limited by this. In fact, this previous limitation is a big reason why I chose to experiment with this algorithm.

Data

Raw Data

The raw dataset used in this project mostly comes from Kaggle [3] (see bibliography). A few additional images were added from my own collection yet a few more were downloaded from websites that specifically allowed CC (Creative Commons) usage. In total, the raw dataset contained 792 images. Figure 1 shows a sample of the raw dataset.

Figure 1: Sample of raw data

Preparing the Dataset

The Kaggle dataset only contained labels for 20 [3], so I modified the dataset by dividing it into clean and dirty images. This resulted in 291, and 501 images

of clean and dirty plates, respectively. Another key step I took was to scale all the images. In order to retain as much quality as possible, I obtained the average of all the images and resized that. For images that didn't scale perfectly, I cropped and padded images to meet obtain homogeneity, as can be seen in Figure 2, I call this data the **minimally processed** dataset.

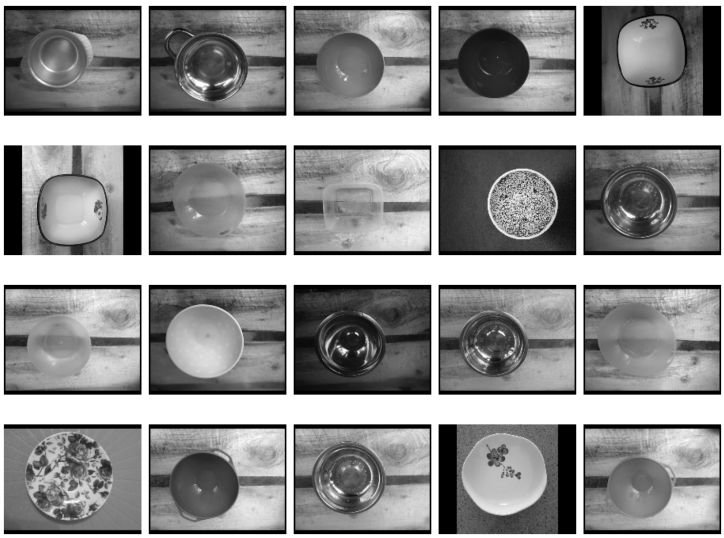

Figure 2: Sample of minimally processed data

Applying SIFT

The next step was to apply SIFT to the minimally processed dataset. Of course, I wanted to compare the performance of the model trained with the minimally processed dataset so I created copies. Before that, I made sure I were applying the algorithm correctly. Figure 3 shows a visual representation of SIFT keypoints after running the algorithm on a raw sample of an image.

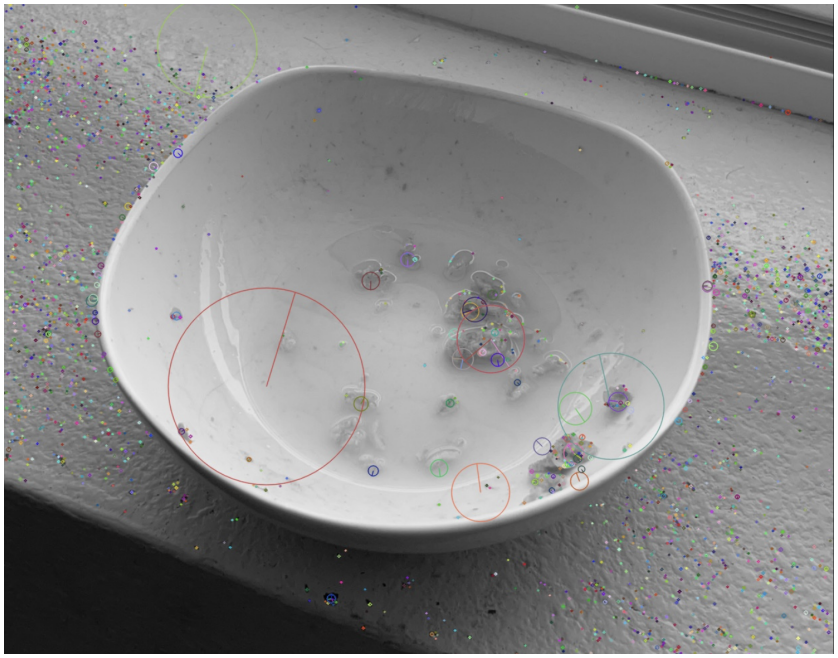

Figure 3: Visual representation of SIFT keypoints

Each keypoint has a direction, position, size, and other features. The keypoints, along with their descriptor vectors are the features I extract from each image. However, I run into an issue very quickly, which is that the homogeneity of the datapoint is now ruined. In fact, this is a known issue with trying to apply SIFT with a CNN model. [2] Prior to this, each image was represented by a 278 x 348 matrix. Now each image is represented by 2 lists, a list of keypoint objects, and a list of descriptor vectors. Both lists are the same length but they vary from image to image, depending on how many SIFT keypoints the algorithm captured. In our data, this varied from 0 to 1163 keypoints. Additionally, each descriptor vector contained 128 elements. I took several steps to solve this issue:

1. Turn every OpenCV keypoint object to a list
2. Delete any datapoints with < 3 keypoints

3. Pad the new keypoint vector with -1 to achieve a length of 128

4. Pad every datapoint with it's last keypoint to achieve a length 1163

This resulted in a new homogeneous dataset with shape of (788, 2, 1163, 128). I call this dataset the SIFT dataset.

Model

Design

The next step in the project was to build 2 models. One to be trained with the minimally processed dataset and the other with the SIFT dataset, I call these model 1 and 2 respectively. As discussed in previous sections, the scope of this project did not include much in the way of thorough experimentation with the CNN architecture. As such, very simple designs were chosen for both models that worked with shapes of their respective datasets.

Model 1

Figure 4 shows the basic design of model 1. As I can see, model 1 is made up of 6 layers, not including the input layer. Note for this project I use the tensorflow keras python implementation of CNN which is very well supported (see bibliography).

I will include the input layer in our summary since it is important for the tensorflow implementation of a CNN.

- **Input Layer:** The network expects an input of shape (278, 348, 1), since our input is a single-channel grayscale image.
- **Layer 1:** A 2D convolutional layer with filters that reduces the spatial dimensions to (276, 346, 3). It outputs feature maps with 3 channels.
- **Layer 2:** A max-pooling layer that halves the spatial dimensions to (138, 173, 3).
- Layer 3: Another 2D convolutional layer that further reduces the spatial dimensions to (136, 171, 3).
- **Layer 4:** A second max-pooling layer reduces the spatial dimensions to (68, 85, 3).

- **Layer 5:** A flattening layer that the 3D tensor into a 1D vector with a total size of 17,340 elements.
- **Layer 6:** A fully connected layer with a single output neuron, producing a scalar output, which can be rounded to obtain a binary.

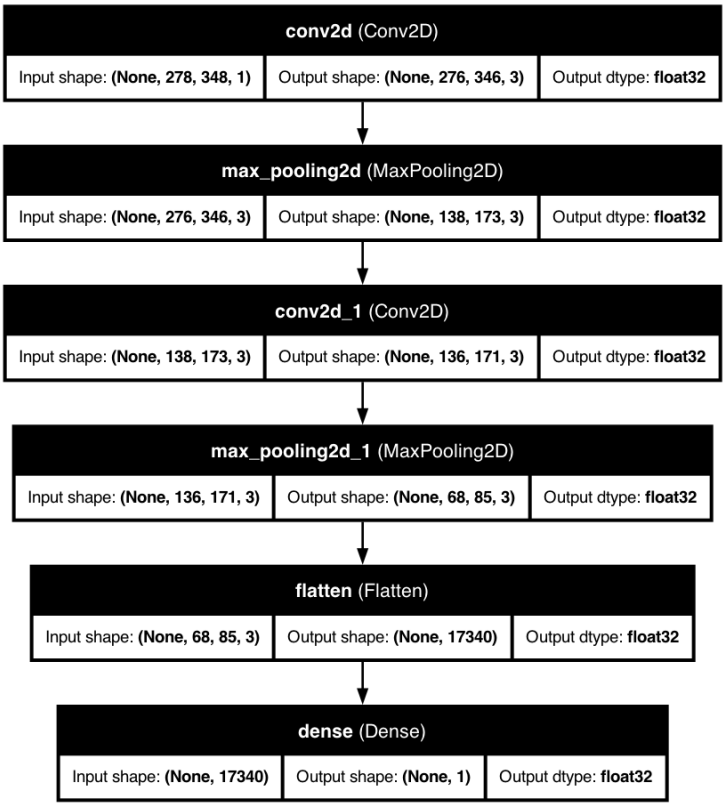

Figure 4: Model 1 Diagram

Model 2

Similarly, model 2 employs a somewhat basic design and an implementation from the tensorflow keras python library. One important note is that the CNN architecture expects the channel dimension to be last in the input data, therefore I must take a crucial step to transpose our data with shape (number of samples, 2, 1163, 128) to (number of samples, 1163, 128, 2). This is a critical step in this application of SIFT, since I can now utilize the 2D architecture of CNN by simply using 2 channel data similar to our grayscale single channel dataset.

1. **Input Layer**: The network expects an input of shape (1163, 128, 2), representing a two-channel image with height 1163 and width 128.
2. **Layer 1**: A 2D convolutional layer that reduces the spatial dimensions to (1161, 126, 32) and outputs 32 feature maps.
3. **Layer 2**: A max-pooling layer that reduces the spatial dimensions by half to (580, 63, 32).
4. **Layer 3**: A second convolutional layer further reduces the dimensions to (578, 61, 64) while increasing the feature maps to 64.
5. **Layer 4**: A second max-pooling layer halves the spatial dimensions again to (289, 30, 64).
6. **Layer 5**: Another convolutional layer reduces the spatial dimensions to (287, 28, 128), increasing the feature maps to 128.
7. **Layer 6**: A third max-pooling layer reduces the spatial dimensions to (143, 14, 128).
8. **Layer 7**: A flattening layer converts the 3D tensor into a 1D vector of size 256,256.
9. **Layer 8**: A fully connected layer with 128 neurons reduces the dimensionality while learning features for classification.
10. **Layer 9**: A dropout layer randomly deactivates some neurons to prevent overfitting, keeping the output size as (128).
11. **Layer 10**: A final fully connected layer with a single output neuron produces a scalar output for binary classification.

Given the size of some dimensions in the SIFT data, I were able to use more layers. I took this as an advantage over using the minimally processed dataset.

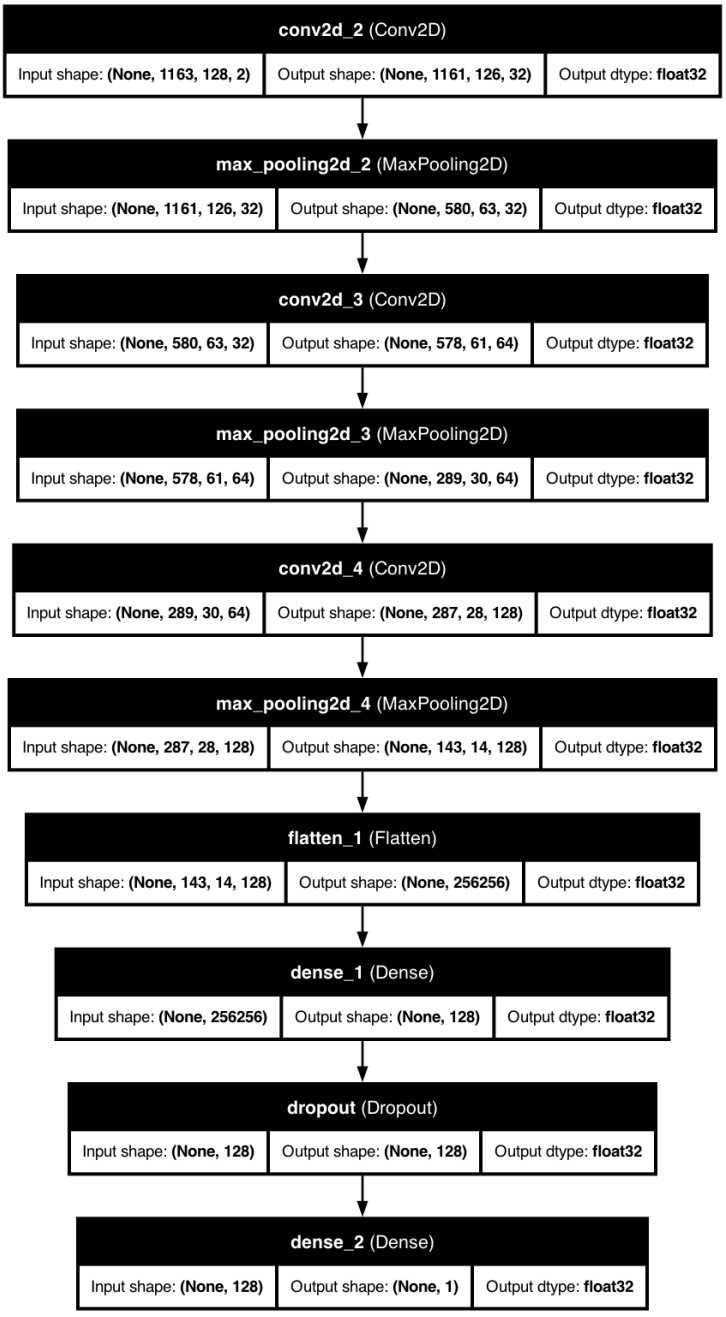

Figure 5: Model 2 Diagram

Model Comparison

A key observation is that the feature extraction, with my padding approach to the homogeneity issue, resulted in a far bigger dataset than with minimal processing. The minimally processed dataset had shape (792, 278, 348), so there were a total of $792 \times 278 \times 348 = 7.7 \times 10^7$ elements. The SIFT dataset had shape (788, 2, 1163, 128), therefore it contained a total of $788 \times 2 \times 1163 \times 128 = 2.3 \times 10^8$) elements.

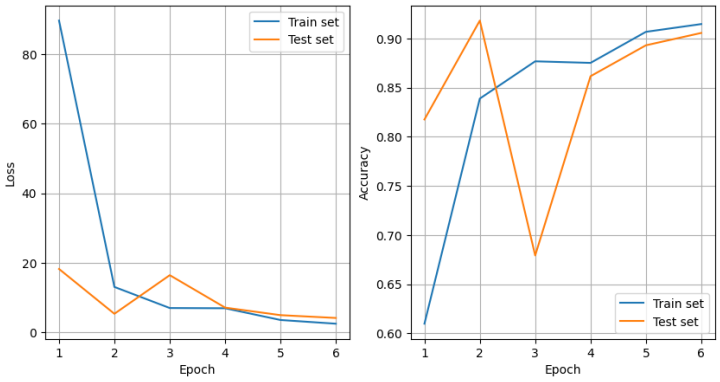

Figure 6: Model 1 Evaluation

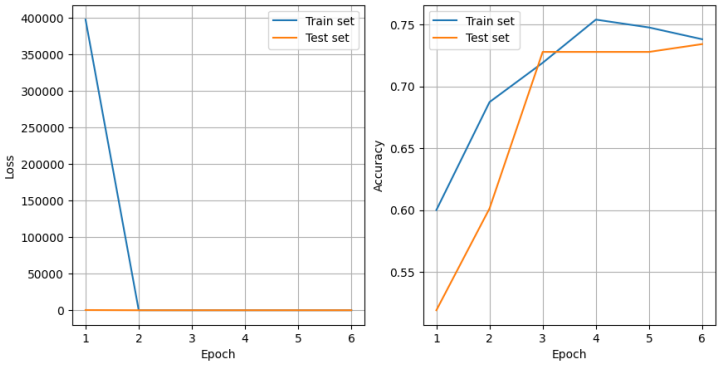

Figure 7: Model 2 Evaluation

This correlates with the training time, where model 1 completed 6 epochs in 19.0 seconds while model 2 completed 6 epochs in 6.3 minutes. Of course the disparity is not entirely due to the size of the training data, since model 2 has nearly twice as many layers than model 1.

Both models used **binary cross-entropy** as the loss function, and **accuracy** as the only metric. Additionally, both models used an 80-20 test-train split of their respective data.

As figures 6 and 7 show, model 1 performed significantly better than model 2! Given that the model without SIFT feature extraction performed better, I chose to use that for our application.

Application

The originally proposed solution was to build a user interface that was easily accessible to anyone needing to use this model to perform classifications. As such, I chose to develop a web application, as shown in Figure 8, for users to easily upload an image of a dish, and have a classification instantly displayed. Of course, in a real application, the back-end would be queried by a robotic interface.

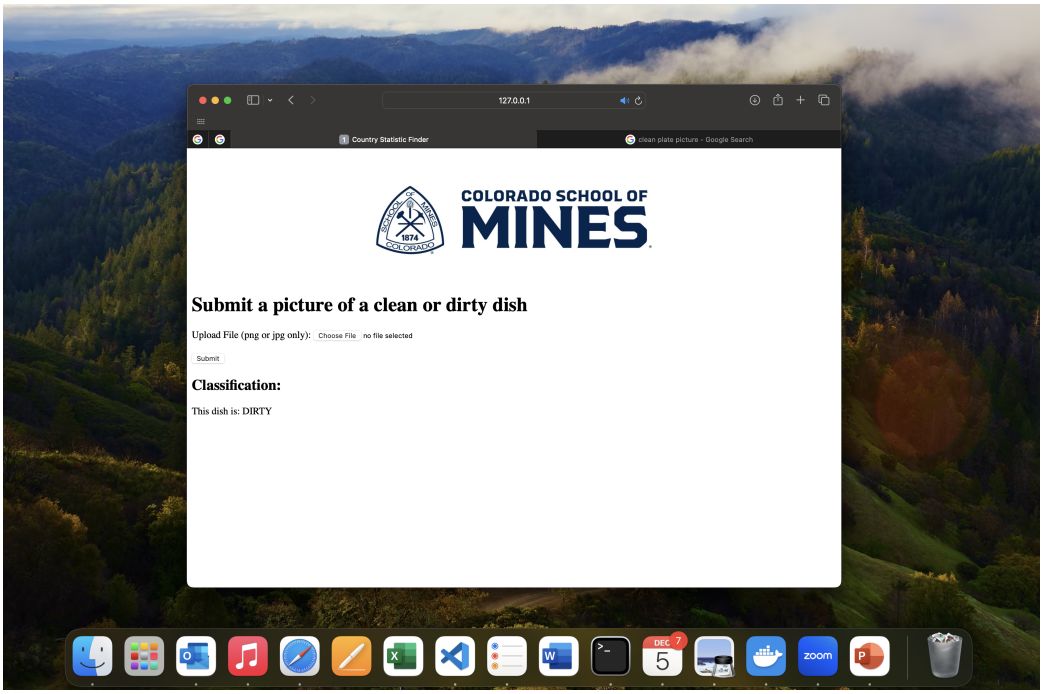

Figure 8: Web App interface to model after receiving an image

For a complete video demonstration click **here**.

The overall design of the interface follows a basic flow, as is shown in

Figure 9. The initial step is to create a local server using the flask python API. Once that is set up, the server waits for a user HTTP GET or POST request.

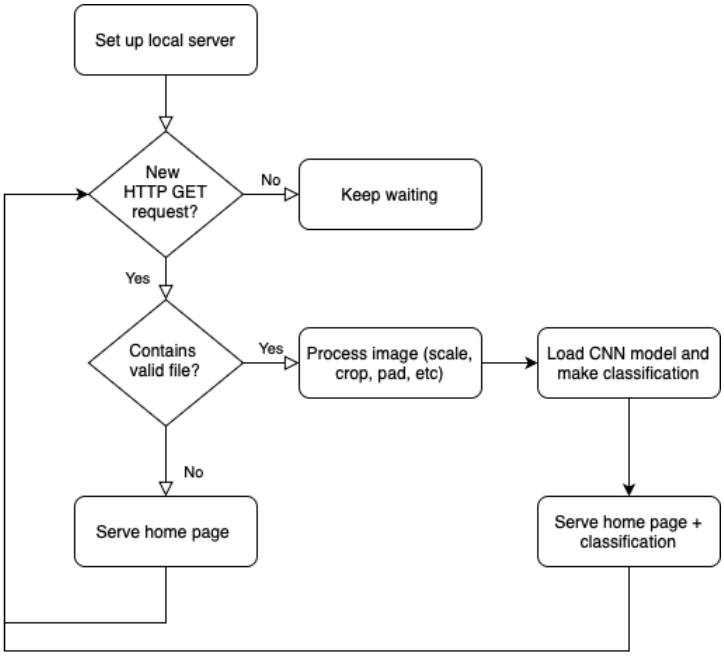

Figure 9: Program flowchart of model interface web app

Of course, most users will have their request packets made by a browser when they enter the localhost address followed by the server port number. When the server receives a request, if the request contains a valid file, then it proceeds to process the image. Recall that the chosen CNN model is trained on minimally processed images of size 348 x 274 (note: the previously referenced matrix shape will be flipped 274 x 348 since python uses column major indexing). This means that the users image needs to be processed in the same way the training data was processed (ie. scale, crop, pad, convert to grayscale) before prompting the model for a prediction. Once the server has processed the image and obtained a prediction, it adds it to the HTML (Hyper Text Markup Language) response and the users screen is updated on their end by their browser, which results in a screen with a classification as shown in Figure 8.

Bibliography

[1] OpenCV. *SIFT Functions from OpenCV*. Available at: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html. Accessed: November 16, 2024.

[2] Dimitrios Tsourounis. *SIFT-CNN Research Paper*. MDPI, 2022. Available at: https://www.mdpi.com/2313-433X/8/10/256. Accessed: November 18, 2024.

[3] Igor Slinko. *Idea and Dataset for PlatesV2*. Kaggle. Available at: https://www.kaggle.com/competitions/platesv2. Accessed: November 16, 2024.

[4] Wikipedia. *Convolutional Neural Network*. Available at: https://en.wikipedia.org/wiki/Convolutional_neural_network. November 25, 2024.

[5] IBM. *Convolutional Neural Networks*. Available at: https://www.ibm.com/topics/convolutional-neural-networks. Accessed: November 29, 2024.