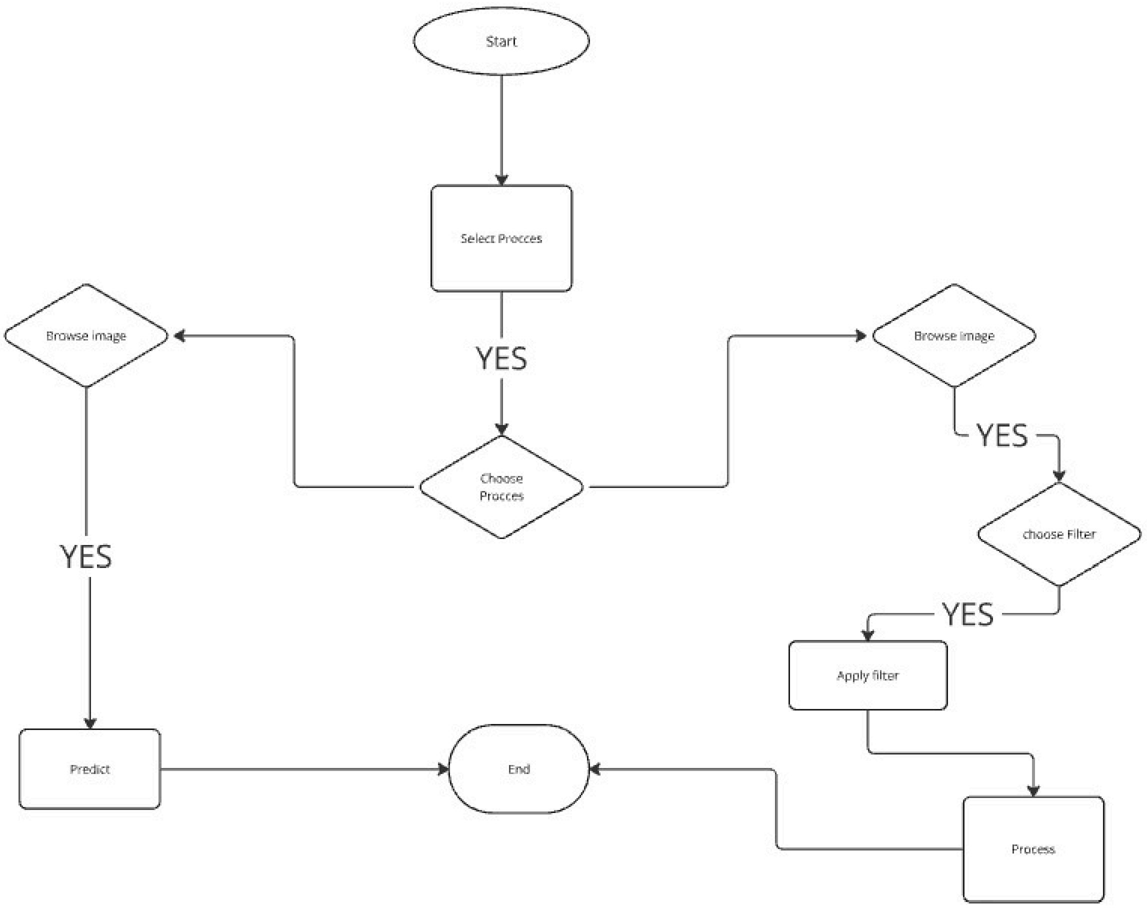
SVM & noise reduction Project Documentation

> **project Name:** Image classification using SVM + noise reduction
>
> **Professor: Dr. Ghada Eltaweel**
>
> **Course:** Digital image processing
>
> **Phase:** Practical Project

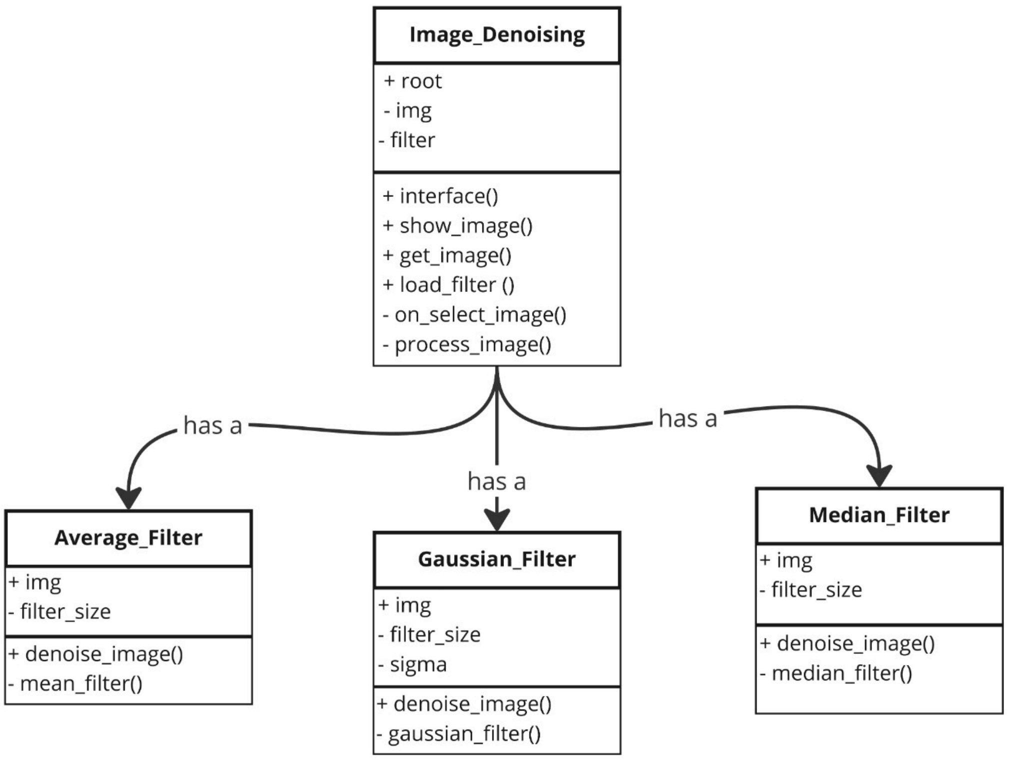
Product Overview 👀

This project focuses on developing a trained image classification model using **Support Vector Machines (SVM)** and enhancing image quality through **noise reduction techniques**. The system classifies images of pets (cats and dogs) by extracting features from grayscale images, training an SVM model, and evaluating its performance, and to handle noisy or low-quality images, the project incorporates **average, Gaussian, and median filters** to preprocess and improve the data.



Start
Select Procces
YES
Choose Procces
Browse image
YES
Predict
Browse image
YES
choose Filter
YES
Apply filter
Process
End

ERD



Image_Denoising
+ root
- img
- filter
+ interface()
+ show_image()
+ get_image()
+ load_filter ()
- on_select_image()
- process_image()
has a
has a
has a
Average_Filter
+ img
- filter_size
+ denoise_image()
- mean_filter()
Gaussian_Filter
+ img
- filter_size
- sigma
+ denoise_image()
- gaussian_filter()
Median_Filter
+ img
- filter_size
+ denoise_image()
- median_filter()

Product Objectives 🎯

1. **Accurate Image Classification**:
   - Train and deploy an SVM-based model to classify images of pets (cats and dogs) effectively.
   - Improve model accuracy through proper data preprocessing, feature normalization, and hyperparameter tuning.
2. **Noise Reduction**:
   - Implement noise reduction filters (average, Gaussian, and median) to preprocess images and enhance their quality.
   - Compare the effectiveness of each filter in reducing noise and its impact on classification performance.
3. **Flexibility and Usability**:
   - Allow users to test the model with new images by specifying their file paths.
   - Provide clear documentation and visualizations of classification predictions and noise reduction results.

Product Features 🧩

1. **Image Preprocessing**:
   - Resizes input images to a uniform size of 50 x 50 pixels.
   - Applies noise reduction using the following filters:
     - **Average Filter**: Reduces noise by averaging pixel intensities in a defined kernel.
     - **Gaussian Filter**: Removes noise by applying a Gaussian kernel for smooth intensity transitions.
     - **Median Filter**: Eliminates salt-and-pepper noise by replacing pixel values with the median of surrounding pixels.
2. **SVM-Based Classification**:
   - Converts images to grayscale to reduce complexity.
   - Employs a Support Vector Machine with hyperparameter optimization using GridSearchCV.
   - Supports multiple kernels, including **linear**, **polynomial**, and **RBF**, for flexible classification.

3. **Dataset Management**:
    - Enables manual control of training and testing datasets by organizing images into `train` and `test` folders.
    - Normalizes pixel intensity values to improve model performance.
4. **Prediction System**:
    - Allows the classification of individual images via a custom prediction function.
    - Provides visual feedback by displaying the input image alongside its predicted category.
5. **Evaluation Metrics**:
    - Calculates accuracy, confusion matrix, and classification report for a comprehensive evaluation of the model's performance.
6. **Flexibility**:
    - The system is scalable to include additional filters or categories.
    - Enables reproducibility by saving and loading the trained SVM model.
7. **Visualization**:
    - Displays test images along with their predicted categories.

First: How to use it:

1. first step: go to the **Model_train** module.
2. second: Put your dataset directory path and initialize your categories like this:

```
train_dir = 'Image_Classification_App/Dataset/train'
categories = ['cats','dogs']
```

3. Use the **load_images_from_folder()** method to store your data
4. Use numpy to process your data for training in this way:

```
xtrain = np.array([item[0] for item in train_data]) / 255.0
ytrain = np.array([item[1] for item in train_data])
```

5. Train your model (mentions below here): 🛠️ SVM & noise reduction Project Documentation
6. save your model in a '.sav' file using pickle
7. Go to the **Prediction** module and put your testing dataset directory for testing

8. Prepare and store your data as mentioned before
9. Open the .sav trained model file
10. prepare your data for testing (use the **predict_image** method) all mentioned below 👇: 🛠 SVM & noise reduction Project Documentation 🛠 SVM & noise reduction Project Documentation

Modules used in the project

they are modules (libraries) used in the Image classification part

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import pickle
import random
from sklearn.model_selection import 
train_test_split,GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,
classification_report
```

Get data ready for training and testing

This method get data (images) from the directory and store it

```
def load_images_from_folder(folder):
    data = []
    path = []
    label =[]

    for category in categories:
        path = os.path.join(folder,category)
        label=categories.index(category)

        for img in os.listdir(path):
            imgpath = os.path.join(path,img)
            try:
```

```python
                pet_img=cv2.imread(imgpath,0)
                pet_img=cv2.resize(pet_img,(50,50))
                image = np.array(pet_img).flatten()
                data.append([image,label])
            except Exception as e:
                pass
    return data
```

Training part

Train the model and get it ready using SVM and save the model

```python
train_data = load_images_from_folder(train_dir)
random.shuffle(train_data)

xtrain = np.array([item[0] for item in train_data]) / 255.0  # Normalize
ytrain = np.array([item[1] for item in train_data])

param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'poly', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=0)
grid.fit(xtrain, ytrain)

model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)


pick = open('model.sav','wb')
pickle.dump(model,pick)
pick.close()
```

Testing

Test the model with the Selected image and show confusion matrix and classification report

```python
pick = open('model.sav','rb')
model = pickle.load(pick)
pick.close()

predictions = model.predict(xtest)
accuracy = model.score(xtest, ytest)
print("Accuracy:", accuracy)

conf_matrix = confusion_matrix(ytest, predictions)
class_report = classification_report(ytest, predictions, target_names=categories)

print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

Method used for testing

```python
def predict_image(image_path, model, categories):
    try:
        pet_img = cv2.imread(image_path, 0)  # Load image in grayscale
        pet_img = cv2.resize(pet_img, (50, 50))
        image = np.array(pet_img).flatten() / 255.0
        prediction = model.predict([image])[0]
        print(f"Prediction for '{image_path}': {categories[prediction]}")

        # Show the image
        plt.imshow(pet_img, cmap='gray')
        plt.title(f"Prediction: {categories[prediction]}")
        plt.show()
    except Exception as e:
        print(f"Error processing image {image_path}: {e}")
```

For the Noise reduction algorithms:

Median Filter

Purpose: Primarily used to remove salt-and-pepper noise from images.

Operation:

- The pixel value is replaced with the median of the values in its neighborhood.
- This is a non-linear filter.

Strengths:

- Preserves edges effectively since it replaces outliers without blurring.
- Robust against impulse noise.

Weaknesses:

- Computationally intensive compared to linear filters.
- Not ideal for Gaussian noise.

Average Filter (Mean Filter)

Purpose: Used for general smoothing of images to reduce noise.

Operation:

- The pixel value is replaced with the average of the values in its neighborhood.
- This is a linear filter.

Strengths:

- Simple and fast to compute.
- Reduces random noise effectively.

Weaknesses:

- Blurs edges and details.
- Not effective at removing high-intensity noise like salt-and-pepper.
-

Gaussian Filter

Purpose: Used for smoothing while preserving important structural details in the image. **Operation:**

- The pixel value is computed using a weighted average where weights follow a Gaussian distribution.
- This is a linear filter.

Strengths:

- Provides a smoother result compared to an average filter.
- Slightly more computationally intensive than the average filter.

Weaknesses:

- Not effective at removing impulse noise like salt-and-pepper noise.
- Not ideal for Gaussian noise.

FAQs 🙋‍♂️

Answer and document frequently asked questions below.

Which algorithm or tool used for classification?

SVM

Which algorithm or tool used for classification?

average, Gaussian, and median filters

Is it a team work?

Yes, for sure.

Program Snapshots 👩‍💻

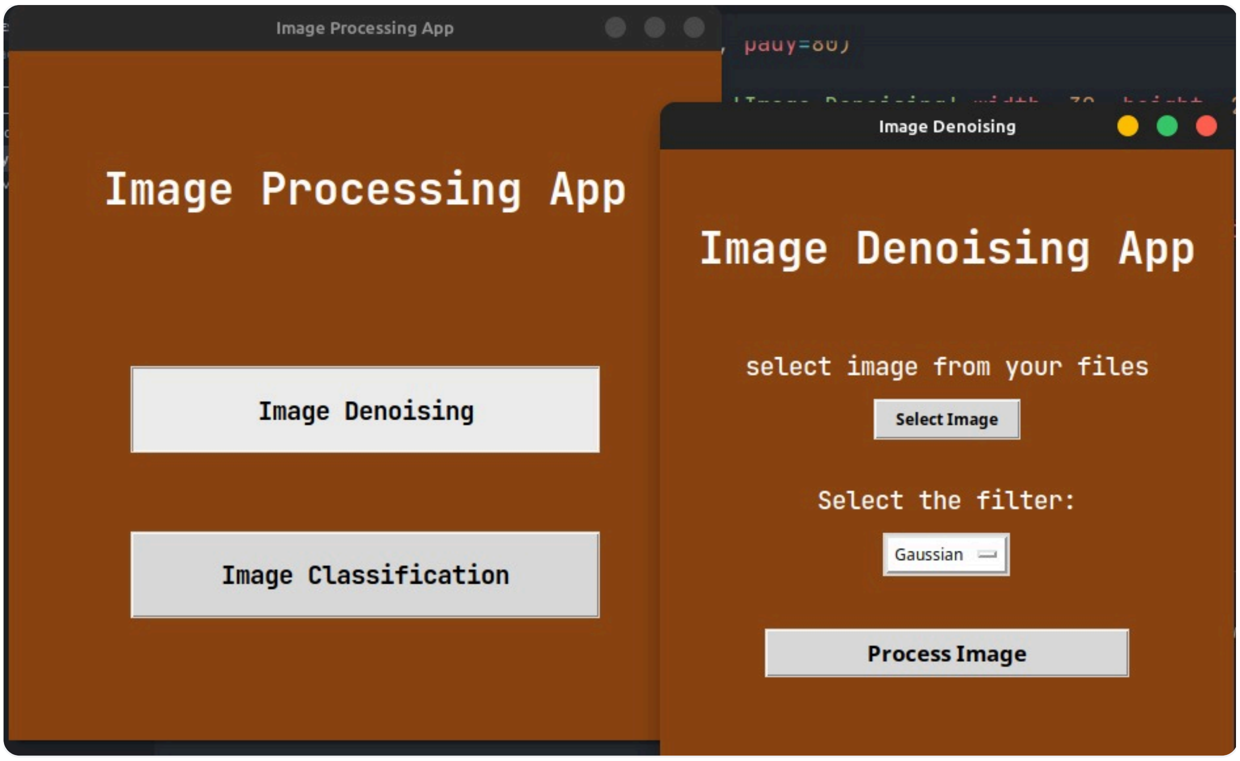


Image Processing App
Image Processing App
Image Denoising
Image Classification
Image Denoising
Image Denoising App
select image from your files
Select Image
Select the filter:
Gaussian
Process Image

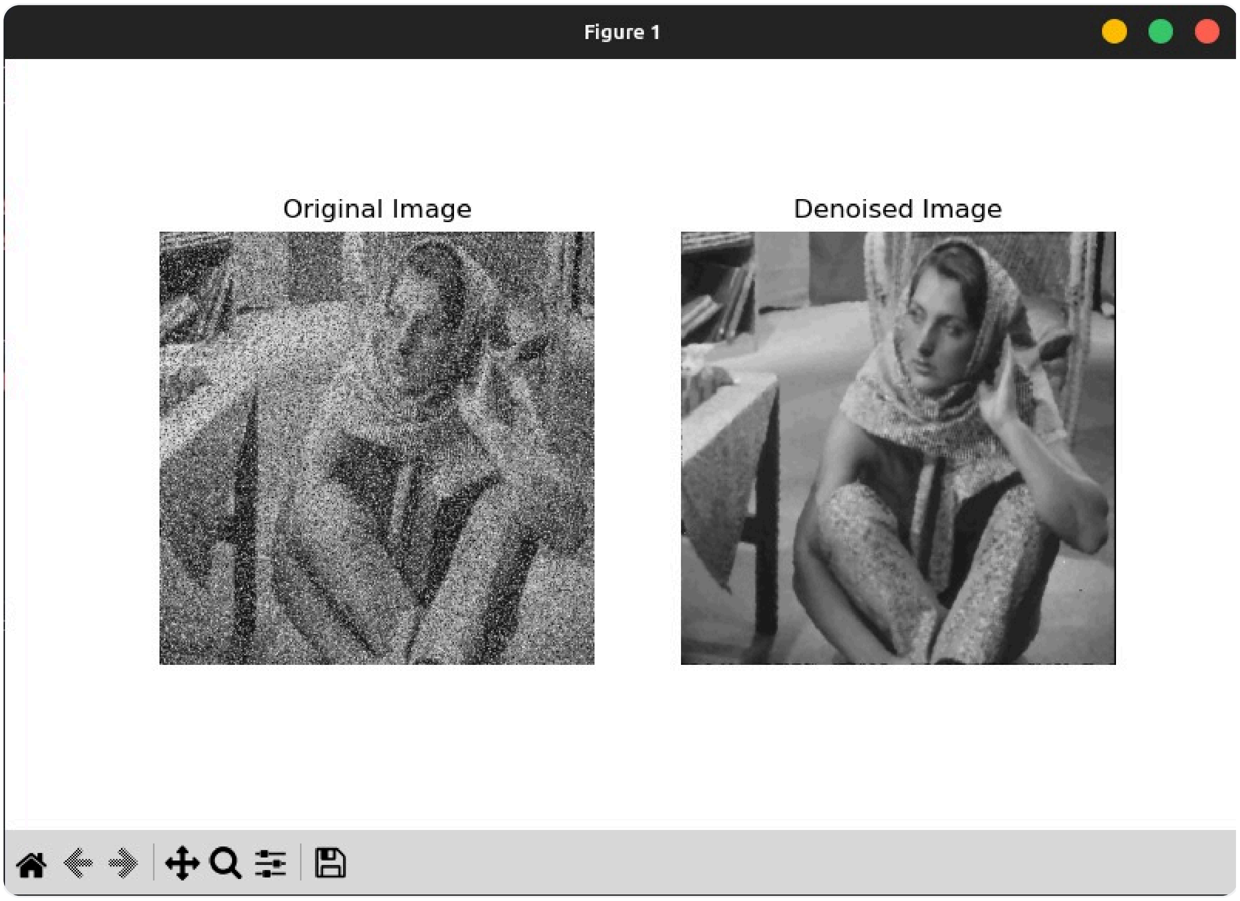
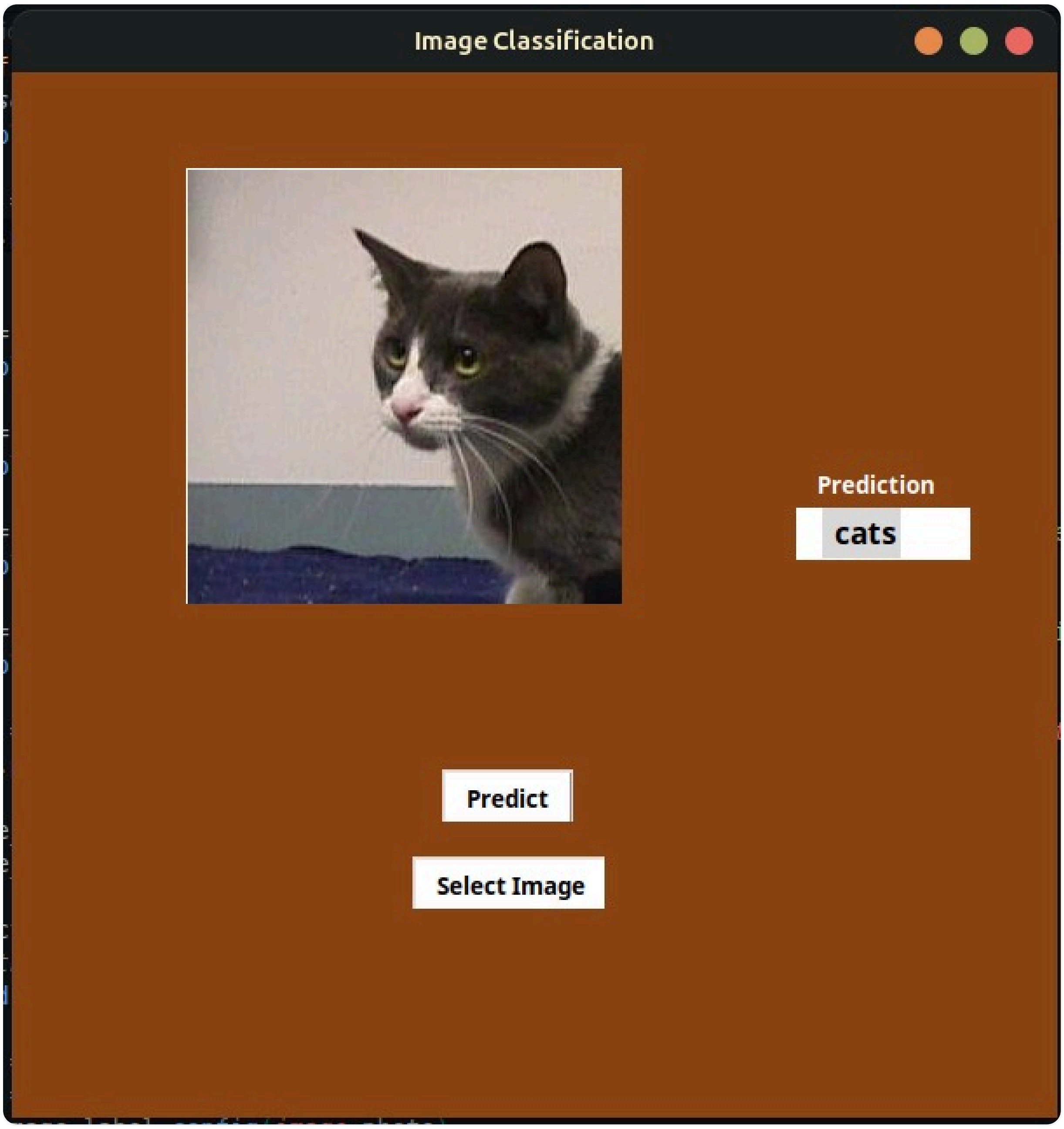


Figure 1
Original Image
Denoised Image



Team members 💬

1. يوسف محمد مسعد الشهاوي (4)
2. محمد حسام احمد فؤاد (3)
3. احمد محمد محمد شوقي بديع عامر (1)
4. محمد نعيم محمد محمد شوشه (3)
5. يوسف هاني محمد السيد محمد صقر (4)
6. محمد احمد صالح عبد العزيز (3)