LABORATORY REPORT

Application Development Lab (CS33002)

B.Tech Program in ECSc

Submitted By

Name:-Pruthibiraj Nayak

Roll No: 2230183

Kalinga Institute of Industrial Technology
(Deemed to be University)
Bhubaneswar, India

Spring 2024-2025

Table of Content

Experiment Number	2
Experiment Title	Machine Learning for Cat and Dog Classification
Date of Experiment	14.01.2025
Date of Submission	20.01.2025

1. Objective:-

To classify images as cats or dogs using machine learning models.

2. Procedure:- (Steps Followed)

1. Collect a labeled dataset of cat and dog images.
2. Preprocess images using OpenCV (resize, flatten, etc.).
3. Train ML models: SVM, Random Forest, Logistic Regression, CNN, and K-means clustering
4. Save the trained models.
5. Build a Flask backend to load models and handle image uploads.
6. Create a frontend with HTML/CSS for uploading images and selecting models.
7. Display the classification result on the webpage.

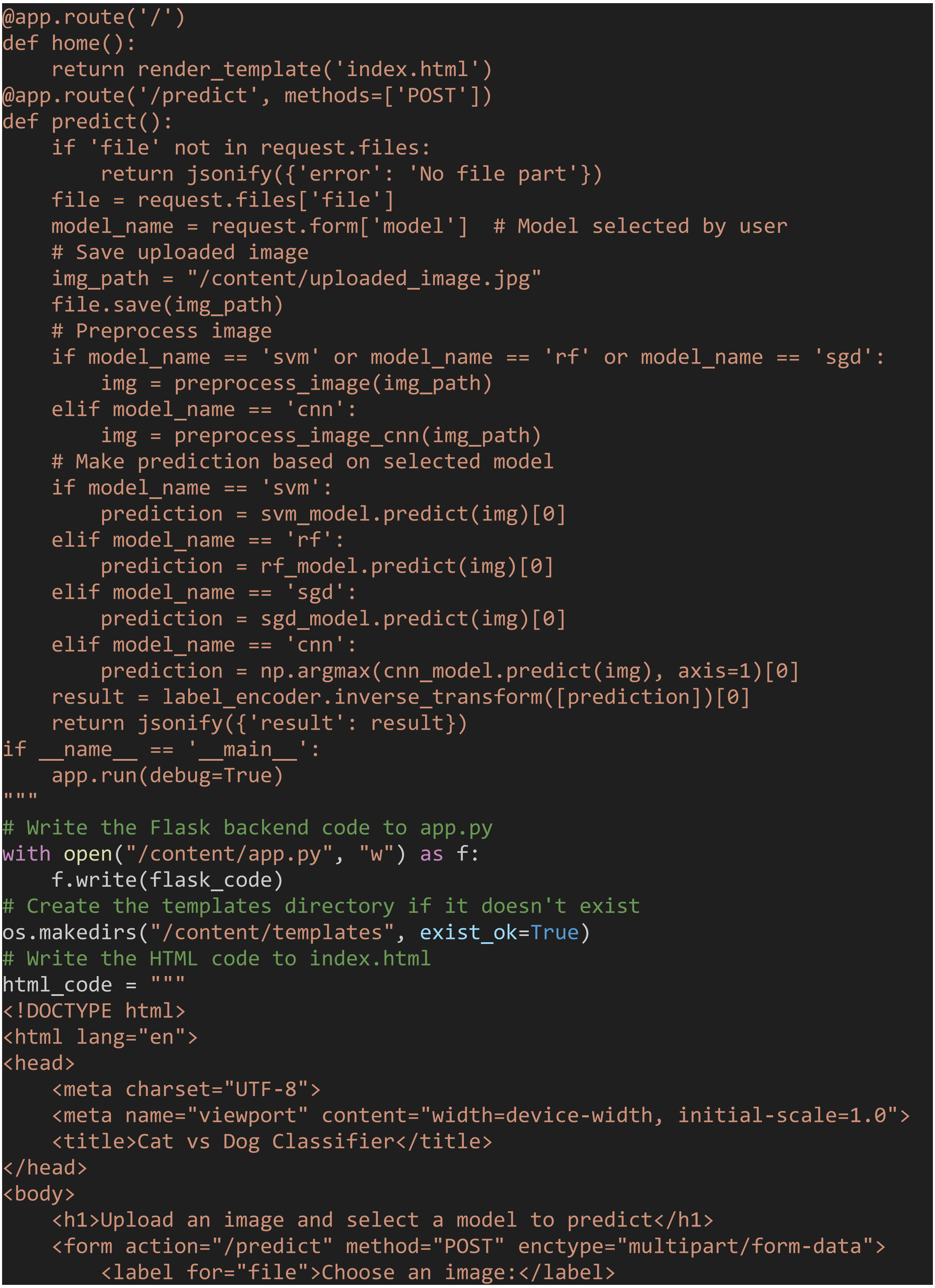
Code:-

```
# -*- coding: utf-8 -*-
"""Untitled15.ipynb

Automatically generated by Colab.
Original file is located at
    https://colab.research.google.com/drive/1H2XUB1zRstnm_fs6aupUxJQUXZipd3
em
"""

import os
import requests
from zipfile import ZipFile
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import joblib
from tensorflow.keras.models import load_model
```

```python
# Download dataset
url = "https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_5340.zip"
dataset_path = "cats_and_dogs.zip"
if not os.path.exists("dataset"):
    print("Downloading dataset...")
    response = requests.get(url)
    with open(dataset_path, 'wb') as file:
        file.write(response.content)
    # Extract dataset
    with ZipFile(dataset_path, 'r') as zip_ref:
        zip_ref.extractall("dataset")

# Preprocess images
def preprocess_image(image_path, size=(16, 16)):  # Reduced size for faster processing
    try:
        image = cv2.imread(image_path)
        image = cv2.resize(image, size)
        image = image / 255.0  # Normalize
        return image
    except:
        return None
def load_data(data_dir, label_map, subset_size=None):
    images, labels = [], []
    for label, folder in label_map.items():
        folder_path = os.path.join(data_dir, folder)
        for i, filename in enumerate(os.listdir(folder_path)):
            if subset_size and i >= subset_size:
                break
            file_path = os.path.join(folder_path, filename)
            image = preprocess_image(file_path)
            if image is not None:
                images.append(image)
                labels.append(label)
    return np.array(images), np.array(labels)

# Load data
data_dir = "dataset/PetImages"
label_map = {0: "Cat", 1: "Dog"}
subset_size = 5000  # Use a subset for faster training
images, labels = load_data(data_dir, label_map, subset_size=subset_size)
# Flatten images for ML models (non-CNN models)
flattened_images = images.reshape(len(images), -1)
# Encode labels
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)
y_categorical = to_categorical(encoded_labels)
```

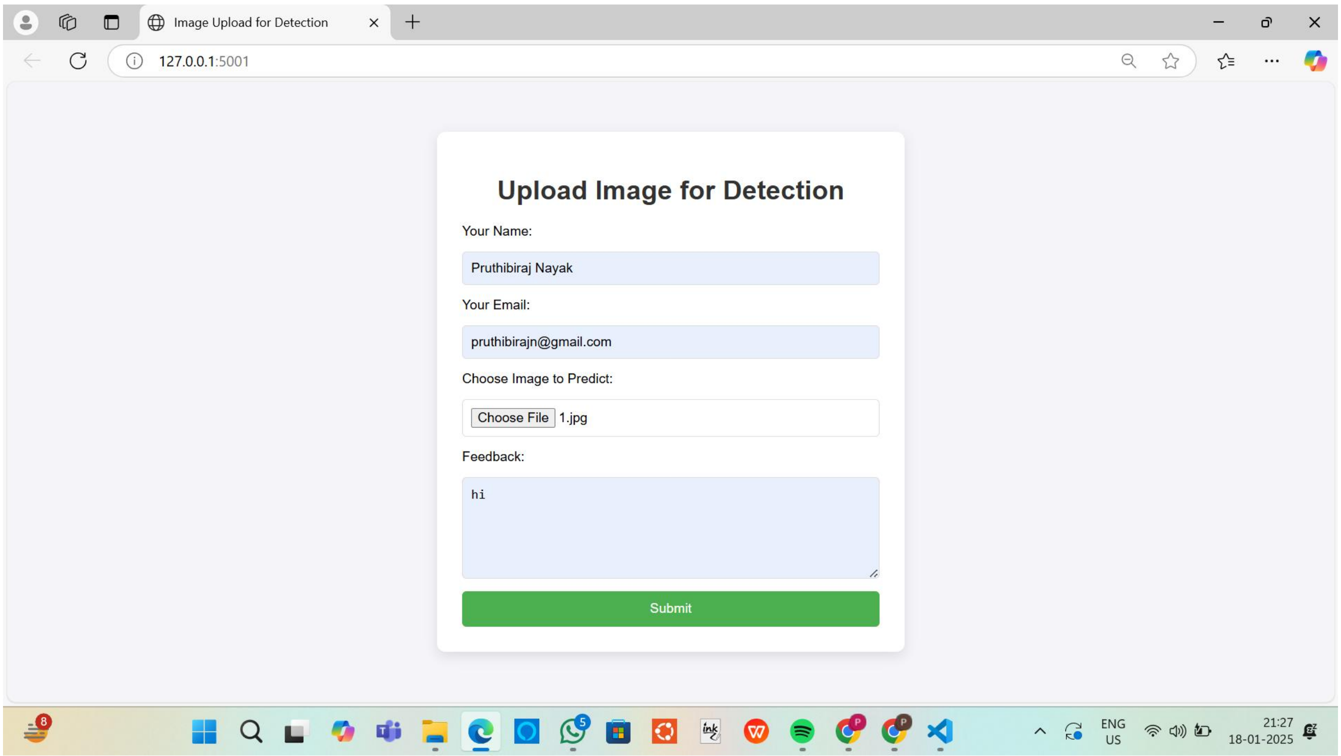
```python
# Split data
X_train, X_test, y_train, y_test = train_test_split(flattened_images, encoded_labels, test_size=0.2, random_state=42)
cnn_X_train, cnn_X_test, cnn_y_train, cnn_y_test = train_test_split(images, y_categorical, test_size=0.2, random_state=42)

# Train SVM
print("Training SVM...")
svm_model = SVC(kernel='linear', C=0.1, probability=True)
svm_model.fit(X_train, y_train)
joblib.dump(svm_model, "svm_model.pkl")
print("SVM training completed and saved.")

# Train Random Forest
print("Training Random Forest...")
rf_model = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train)
joblib.dump(rf_model, "rf_model.pkl")
print("Random Forest training completed and saved.")

# Train Logistic Regression (SGD)
print("Training Logistic Regression...")
sgd_model = SGDClassifier(loss='log_loss', max_iter=1000, random_state=42)  # Updated loss parameter
sgd_model.fit(X_train, y_train)
joblib.dump(sgd_model, "sgd_model.pkl")
print("Logistic Regression training completed and saved.")

# Train CNN
print("Training CNN...")
cnn_model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=(16, 16, 3)),  # Fewer filters
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),  # Smaller dense layer
    Dense(2, activation='softmax')
])

cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
cnn_model.fit(cnn_X_train, cnn_y_train, epochs=30, batch_size=64, validation_data=(cnn_X_test, cnn_y_test))  # Fewer epochs
cnn_model.save("cnn_model.h5")
print("CNN training completed and saved.")

# Load models for inference
print("Loading models for inference...")
```

```
svm_model = joblib.load("svm_model.pkl")
rf_model = joblib.load("rf_model.pkl")
sgd_model = joblib.load("sgd_model.pkl")
cnn_model = load_model("cnn_model.h5")

# Test on one sample image
sample_image = X_test[0].reshape(1, -1)  # For non-CNN models
cnn_sample_image = cnn_X_test[0].reshape(1, 16, 16, 3)  # For CNN
print("SVM Prediction:",
label_encoder.inverse_transform(svm_model.predict(sample_image)))
print("Random Forest Prediction:",
label_encoder.inverse_transform(rf_model.predict(sample_image)))
print("Logistic Regression Prediction:",
label_encoder.inverse_transform(sgd_model.predict(sample_image)))
print("CNN Prediction:",
label_encoder.inverse_transform(np.argmax(cnn_model.predict(cnn_sample_imag
e), axis=1)))

!pip install flask pyngrok
# Step 2: Flask Backend Code (Write to app.py)
flask_code = """
from flask import Flask, request, jsonify, render_template
import numpy as np
import cv2
import joblib
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from sklearn.preprocessing import LabelEncoder
app = Flask(__name__)
# Load models
svm_model = joblib.load("svm_model.pkl")
rf_model = joblib.load("rf_model.pkl")
sgd_model = joblib.load("sgd_model.pkl")
cnn_model = load_model("cnn_model.h5")
# Label encoder
label_encoder = LabelEncoder()
label_encoder.fit(["Cat", "Dog"])
# Preprocess image for non-CNN models
def preprocess_image(image_path, size=(16, 16)):
    img = cv2.imread(image_path)
    img = cv2.resize(img, size)
    img = img / 255.0  # Normalize
    return img.reshape(1, -1)  # Flatten for SVM, RF, SGD
# Preprocess image for CNN model
def preprocess_image_cnn(image_path, size=(16, 16)):
    img = cv2.imread(image_path)
    img = cv2.resize(img, size)
    img = img / 255.0  # Normalize
    return img.reshape(1, 16, 16, 3)  # Reshape for CNN
```



```
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'})
    file = request.files['file']
    model_name = request.form['model']  # Model selected by user
    # Save uploaded image
    img_path = "/content/uploaded_image.jpg"
    file.save(img_path)
    # Preprocess image
    if model_name == 'svm' or model_name == 'rf' or model_name == 'sgd':
        img = preprocess_image(img_path)
    elif model_name == 'cnn':
        img = preprocess_image_cnn(img_path)
    # Make prediction based on selected model
    if model_name == 'svm':
        prediction = svm_model.predict(img)[0]
    elif model_name == 'rf':
        prediction = rf_model.predict(img)[0]
    elif model_name == 'sgd':
        prediction = sgd_model.predict(img)[0]
    elif model_name == 'cnn':
        prediction = np.argmax(cnn_model.predict(img), axis=1)[0]
    result = label_encoder.inverse_transform([prediction])[0]
    return jsonify({'result': result})
if __name__ == '__main__':
    app.run(debug=True)
"""
# Write the Flask backend code to app.py
with open("/content/app.py", "w") as f:
    f.write(flask_code)
# Create the templates directory if it doesn't exist
os.makedirs("/content/templates", exist_ok=True)
# Write the HTML code to index.html
html_code = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cat vs Dog Classifier</title>
</head>
<body>
    <h1>Upload an image and select a model to predict</h1>
    <form action="/predict" method="POST" enctype="multipart/form-data">
        <label for="file">Choose an image:</label>
        <input type="file" name="file" accept="image/*" required><br><br>
        <label for="model">Select a model:</label>
        <select name="model" required>
            <option value="svm">SVM</option>
            <option value="rf">Random Forest</option>
            <option value="sgd">Logistic Regression (SGD)</option>
            <option value="cnn">CNN</option>
        </select><br><br>
        <input type="submit" value="Predict">
    </form>
    {% if result %}
        <h2>Prediction: {{ result }}</h2>
    {% endif %}
</body>
</html>
"""
# Write the HTML code to index.html
with open("/content/templates/index.html", "w") as f:
    f.write(html_code)

from pyngrok import ngrok
# Replace 'your-authtoken' with the token you copied from your ngrok
dashboard
ngrok.set_auth_token("your-authtoken")
```

3. Results/Output:- Entire Screen Shot including Date & Time



Upload Image for Detection

Your Name:

Pruthibiraj Nayak

Your Email:

pruthibirajn@gmail.com

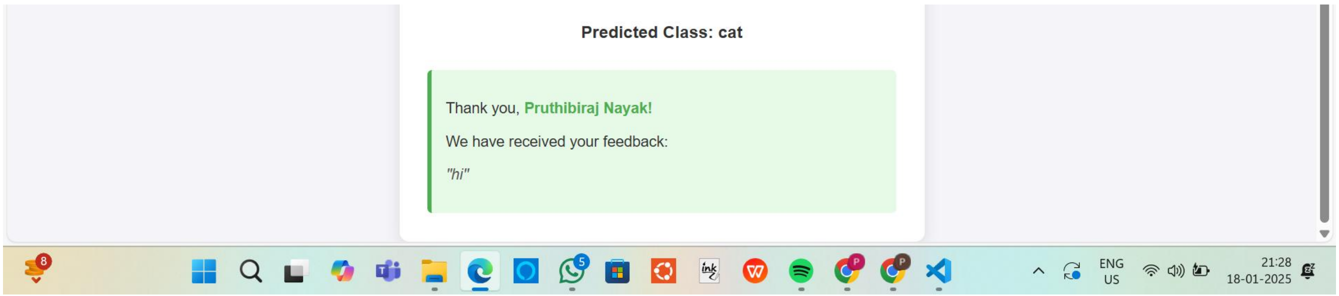
Choose Image to Predict:

Choose File 1.jpg

Feedback:

hi

Submit



Predicted Class: cat

Thank you, Pruthibiraj Nayak!

We have received your feedback:

"hi"

(CNN model Verification)

4. Remarks:-

This experiment successfully classified cat and dog images using machine learning models like CNN while just accomplishing training and testing using SVM, Random Forest, Logistic Regression, and K-means Clustering. The CNN model demonstrated superior performance due to its ability to learn complex image features. The deployment via Flask and a user-friendly HTML/CSS frontend ensured practical usability. This project highlights the effectiveness of integrating machine learning models into web applications for real-world image classification tasks.

Pruthibiraj Nayak (2230183)

(Name of the Student)

(Name of the Coordinator)