LABORATORY REPORT

Application Development Lab
(CS33002)

B.Tech Program in ECSc

Submitted By

Name: Shriya Shukla

Roll No: 2230201

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

Spring 2024-2025

Table of Content

Experiment Number	2
Experiment Title	To design and develop Machine Learning model for Cat and Dog Classification
Date of Experiment	14/01/2025
Date of Submission	20/01/2025

Objective:- To design and develop Machine Learning model for Cat and Dog Classification

Procedure:-

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:-

```
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

# 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)
# 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_image), axis=1)))

# Train K-Means
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')  # Suppress warnings for clean outpu
print("Training K-Means...")
kmeans_model = KMeans(n_clusters=2, random_state=42)
```

```
kmeans_model.fit(X_train)  # Unsupervised training on flattened images
joblib.dump(kmeans_model, "kmeans_model.pkl")
print("K-Means training completed and saved.")

!pip install flask-ngrok flask tensorflow scikit-learn pillow
!pip install jupyter-dash
import plotly.express as px
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output# Load Data
pip install pyngrok

from flask import Flask
from pyngrok import ngrok

ngrok.set_auth_token('2rtA0iFOshOyTXXDlIzIHBwedUK_2PCaFQ7Bb
WdGDScjzNtyo')
public_url = ngrok.connect(5000).public_url
print(public_url)
from flask import Flask, request, jsonify, render_template
from tensorflow.keras.models import load_model
import joblib
import cv2
import numpy as np
from pyngrok import ngrok

# Initialize Flask app
app = Flask(__name__)
```

```
!pkill ngrok

# Set up ngrok
public_url = ngrok.connect(5000)
print(f"Public URL: {public_url}")

# 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")
kmeans_model = joblib.load("kmeans_model.pkl") # Load KMeans model

# Label map
label_map = {0: "Cat", 1: "Dog"}

def inverse_label(label_idx):
    return label_map[label_idx]

# Preprocess image
def preprocess_image(image_file, size=(16, 16)):
    image = cv2.imdecode(np.frombuffer(image_file.read(), np.uint8), cv2.IMREAD_COLOR)
    if image is None:
        return None
    image = cv2.resize(image, size)
    image = image / 255.0  # Normalize
```

```
    return image

# Root route
@app.route('/')
def home():
    return """
    <html>
        <head><title>Cat and Dog Classifier</title></head>
        <body>
            <h1>Welcome to the Cat and   Classifier</h1>
                                <form      action="/predict"      method="post"
enctype="multipart/form-data">
                <label for="image">Upload an image:</label>
                    <input   type="file"   name="image"   accept="image/*"
required>
                <button type="submit">Predict</button>
            </form>
        </body>
    </html>
    """

# Prediction route
@app.route('/predict', methods=['POST'])
def predict():
    if 'image' not in request.files:
        return jsonify({'error': 'No image uploaded'}), 400

    image_file = request.files['image']
    image = preprocess_image(image_file)
```

```
    if image is None:
        return jsonify({'error': 'Invalid image format'}), 400

flattened_image = image.reshape(1, -1)
# CNN requires a 4D tensor
cnn_image = image.reshape(1, 16, 16, 3)
# Make predictions
                                        svm_prediction                  =
inverse_label(svm_model.predict(flattened_image)[0])
    rf_prediction = inverse_label(rf_model.predict(flattened_image)[0])
                                        sgd_prediction                  =
inverse_label(sgd_model.predict(flattened_image)[0])
                                        cnn_prediction                  =
inverse_label(np.argmax(cnn_model.predict(cnn_image), axis=1)[0])

    # KMeans prediction (returns cluster number)
    kmeans_cluster = kmeans_model.predict(flattened_image)[0]
    kmeans_prediction = f"Cluster {kmeans_cluster}"

    return jsonify({
        'svm_prediction': svm_prediction,
        'rf_prediction': rf_prediction,
        'sgd_prediction': sgd_prediction,
        'cnn_prediction': cnn_prediction,
        'kmeans_prediction': kmeans_prediction
    })

if __name__ == '__main__':
```

app.run(port=5000)

Results/Output:-

User Interface:

Output:

Remarks:-

From this experiment I learned how to use machine learning models to classify whether the image given as input by the user is of a cat or dog.

Signature of the Student

Shriya Shukla

Signature of the Lab Coordinator