LABORATORY REPORT

Application Development Lab (CS33002)

B. Tech Program in ECSc

Submitted By

Name:- Akriti Patro

Roll No: 2230232

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	14/01/25
Date of Submission	20/01/25

1. **Objective:-** To classify images as cats or dogs using machine learning models.

2. **Procedure:-**

   1. I collected a labeled dataset of cat and dog images.
   2. I preprocessed the images using OpenCV(resize,flatten,etc.).
   3. I trained ML models: SVM, Random Forest, Logistic Regression, CNN, and K-means Clustering.
   4. I saved the trained models.
   5. I built a Flask backend to load the models and handle image uploads.
   6. I created a frontend with HTML/CSS for uploading images and selecting models.
   7. I displayed the classification result on the webpage.

3. **Code:-**

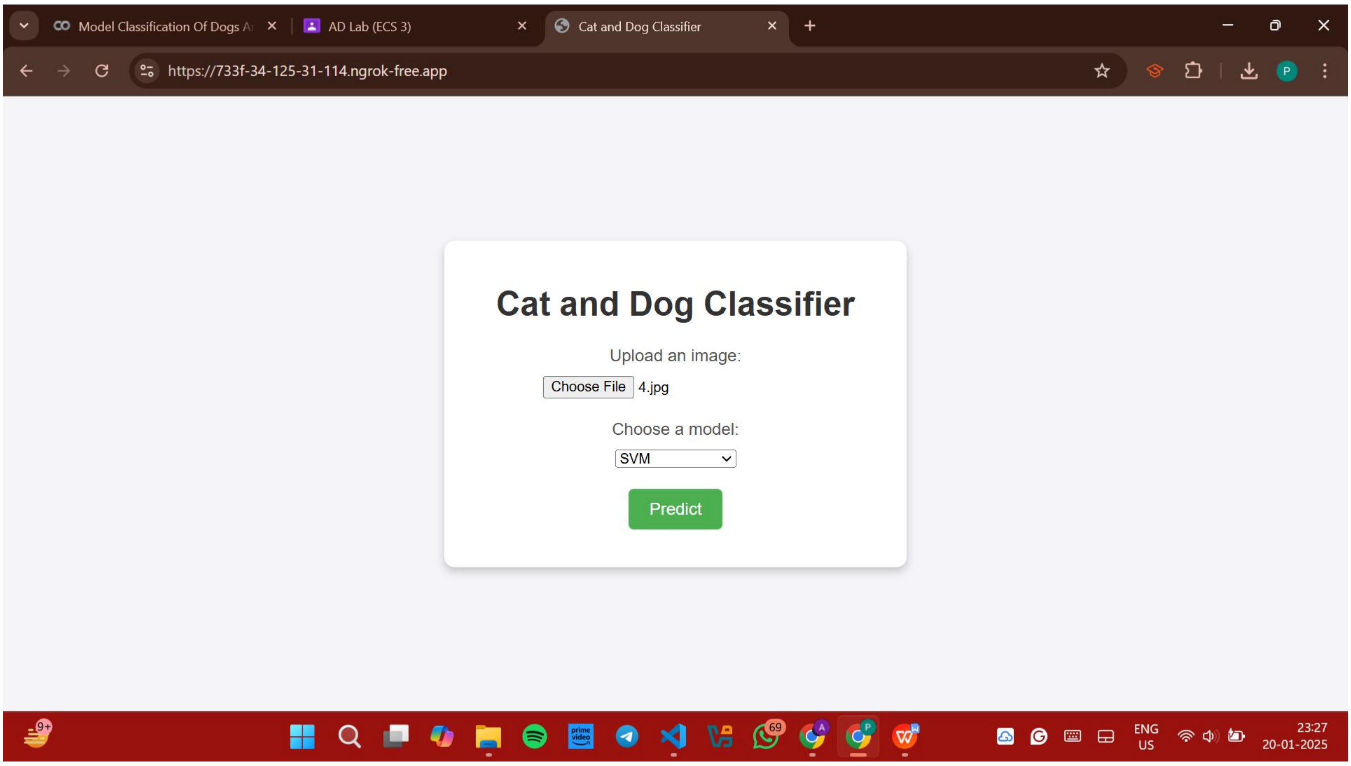
```
import os
import requests
from zipfile import ZipFile
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
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 #1

!pip install jupyter-dash #2
import plotly.express as px
from jupyter_dash import JupyterDash   #3
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output# Load Data
! pip install pyngrok  #4
from flask import Flask #5
from pyngrok import ngrok
ngrok.set_auth_token('2rt5L03UtaVeBd6H3jtAL03eB5A_83J2h8Fb7z8kFxL4cmkWx')
public_url = ngrok.connect(5000).public_url
print(public_url) #6
import os

# Create templates directory
os.makedirs("templates", exist_ok=True)

# Create result.html file inside templates directory
html_content = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prediction Result</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f9;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
        }
        .container {
            text-align: center;
            background: #ffffff;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
            width: 400px;
        }
        h1 {
            color: #333;
        }
        p {
            font-size: 16px;
            color: #555;
        }
        a {
            text-decoration: none;
            color: #4CAF50;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Prediction Result</h1>
        <p>Model Used: {{ model }}</p>
        <p>Prediction: {{ prediction }}</p>
        <a href="/">Go back</a>
    </div>
</body>
</html>
"""

# Write the HTML content to result.html
with open("templates/result.html", "w") as file:
    file.write(html_content)

from flask import Flask, request, jsonify, render_template
import joblib
import cv2
import numpy as np
from pyngrok import ngrok

# Initialize Flask app
app = Flask(__name__)

# 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>
            <style>
                body {
                    font-family: Arial, sans-serif;
                    background-color: #f4f4f9;
                    margin: 0;
                    padding: 0;
                    display: flex;
                    justify-content: center;
                    align-items: center;
                    height: 100vh;
                }
                .container {
                    text-align: center;
                    background: #ffffff;
                    padding: 20px;
                    border-radius: 10px;
                    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
                    width: 400px;
                }
                h1 {
                    color: #333;
                }
                label {
                    font-size: 16px;
                    color: #555;
                }
                input[type="file"], select {
                    margin-top: 10px;
                    margin-bottom: 20px;
                }
                button {
                    background-color: #4CAF50;
                    color: white;
                    border: none;
                    padding: 10px 20px;
                    text-align: center;
                    font-size: 16px;
                    border-radius: 5px;
                    cursor: pointer;
                    transition: background-color 0.3s;
                }
                button:hover {
                    background-color: #45a049;
                }
            </style>
        </head>
        <body>
            <div class="container">
                <h1>Cat and Dog Classifier</h1>
                <form action="/predict" method="post" enctype="multipart/form-data">
                    <label for="image">Upload an image:</label><br>
                    <input type="file" name="image" accept="image/*" required><br>
                    <label for="model">Choose a model:</label><br>
                    <select name="model" required>
                        <option value="svm">SVM</option>
                        <option value="rf">Random Forest</option>
                        <option value="sgd">SGD</option>
                        <option value="cnn">CNN</option>
                        <option value="kmeans">KMeans</option>
                    </select><br>
                    <button type="submit">Predict</button>
                </form>
            </div>
        </body>
    </html>
    """

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

        if 'model' not in request.form:
            return jsonify({'error': 'No model selected'}), 400

        image_file = request.files['image']
        selected_model = request.form['model']
        image = preprocess_image(image_file)

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

        # Flatten image for non-CNN models
        flattened_image = image.reshape(1, -1)

        # CNN requires a 4D tensor
        cnn_image = image.reshape(1, 16, 16, 3)

        # Make prediction based on selected model
        if selected_model == "svm":
            prediction = inverse_label(svm_model.predict(flattened_image)[0])
        elif selected_model == "rf":
            prediction = inverse_label(rf_model.predict(flattened_image)[0])
        elif selected_model == "sgd":
            prediction = inverse_label(sgd_model.predict(flattened_image)[0])
        elif selected_model == "cnn":
            prediction = inverse_label(np.argmax(cnn_model.predict(cnn_image), axis=1)[0])
        elif selected_model == "kmeans":
            cluster = kmeans_model.predict(flattened_image)[0]
            prediction = f"Cluster {cluster}"
        else:
            return jsonify({'error': 'Invalid model selected'}), 400

        return render_template('result.html', model=selected_model, prediction=prediction)

    if __name__ == '__main__':
        app.run(port=5000)

    ### 0: "Cat", 1: "Dog"
```

4. Results/Output:-


Model Classification Of Dogs A
AD Lab (ECS 3)
Cat and Dog Classifier
https://733f-34-125-31-114.ngrok-free.app
ENG
US
23:27
20-01-2025

Cat and Dog Classifier

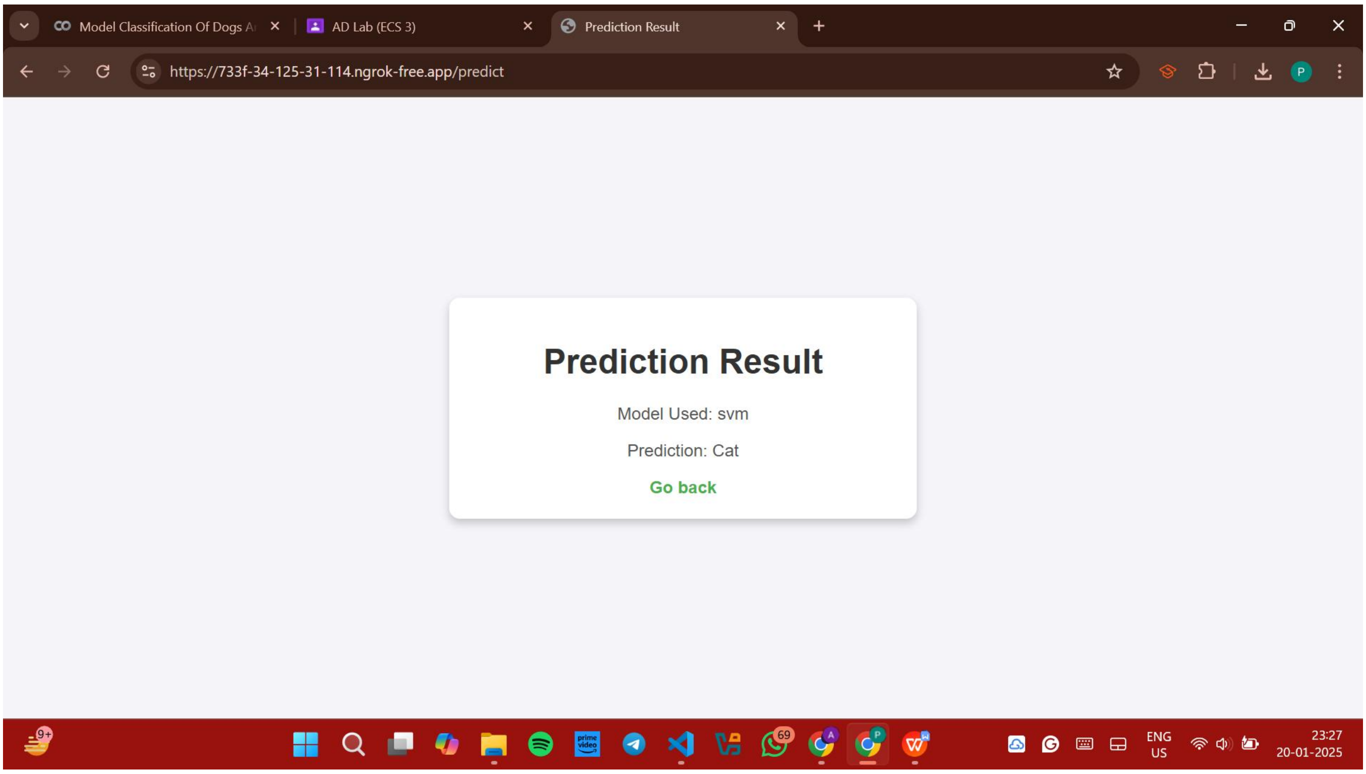
Upload an image:

Choose File 4.jpg

Choose a model:

SVM

Predict



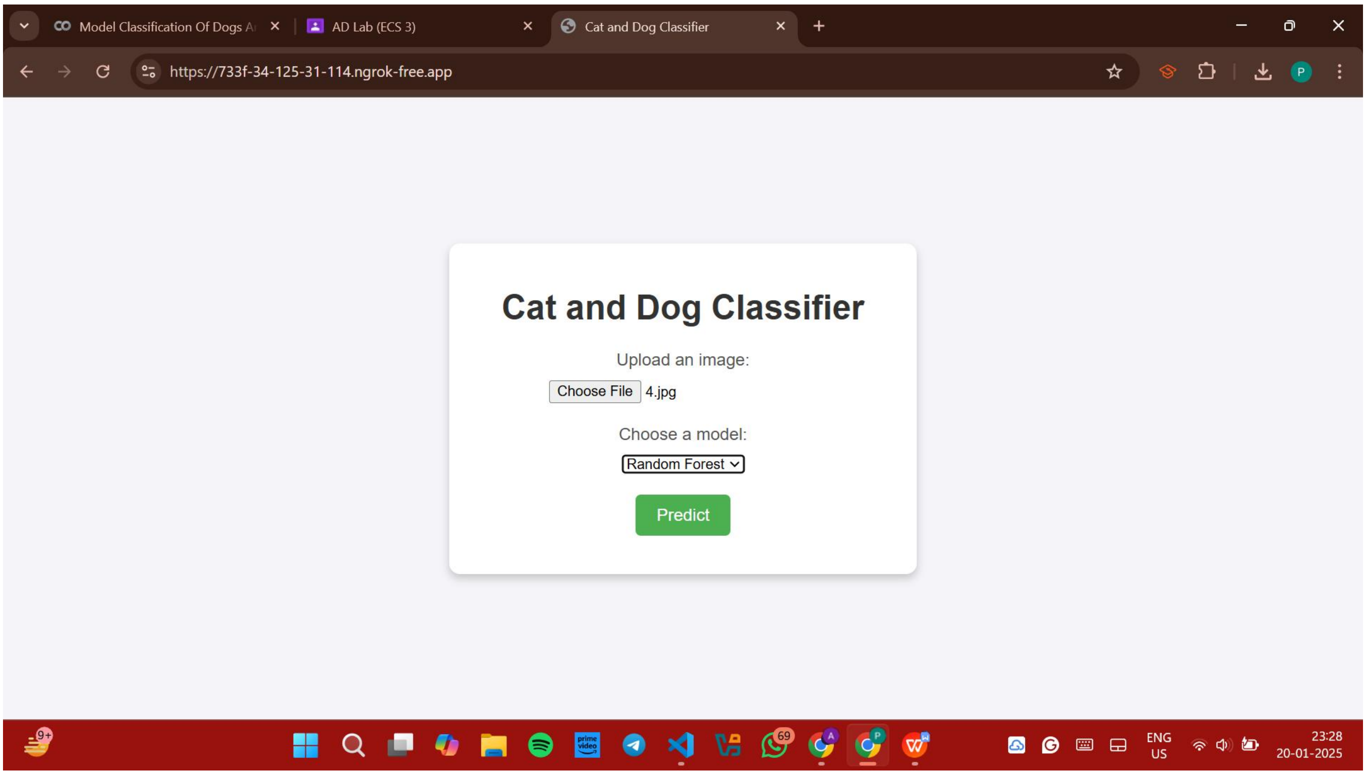
Model Classification Of Dogs A
AD Lab (ECS 3)
Prediction Result
https://733f-34-125-31-114.ngrok-free.app/predict

Prediction Result

Model Used: svm

Prediction: Cat

Go back



Model Classification Of Dogs A
AD Lab (ECS 3)
Cat and Dog Classifier
https://733f-34-125-31-114.ngrok-free.app

Cat and Dog Classifier

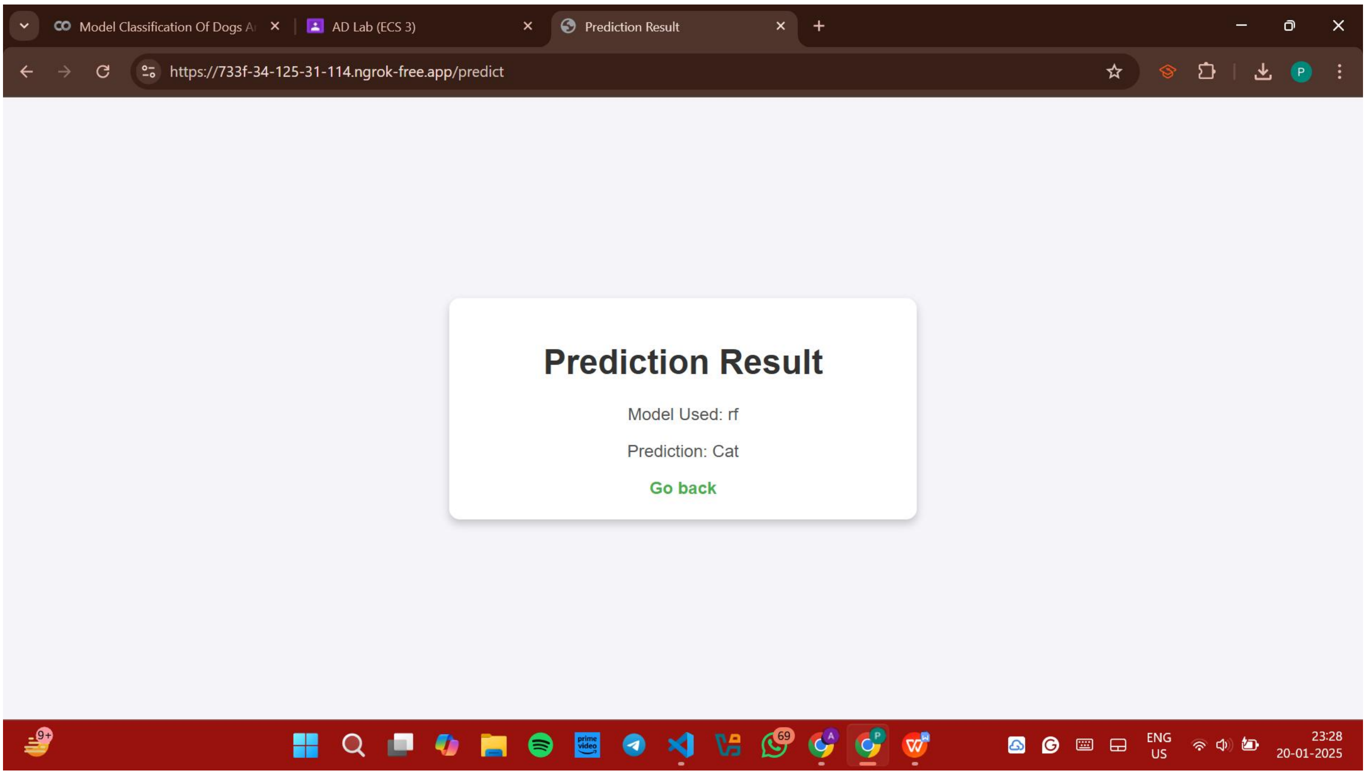
Upload an image:

Choose File 4.jpg

Choose a model:

Random Forest

Predict
ENG
US
23:28
20-01-2025

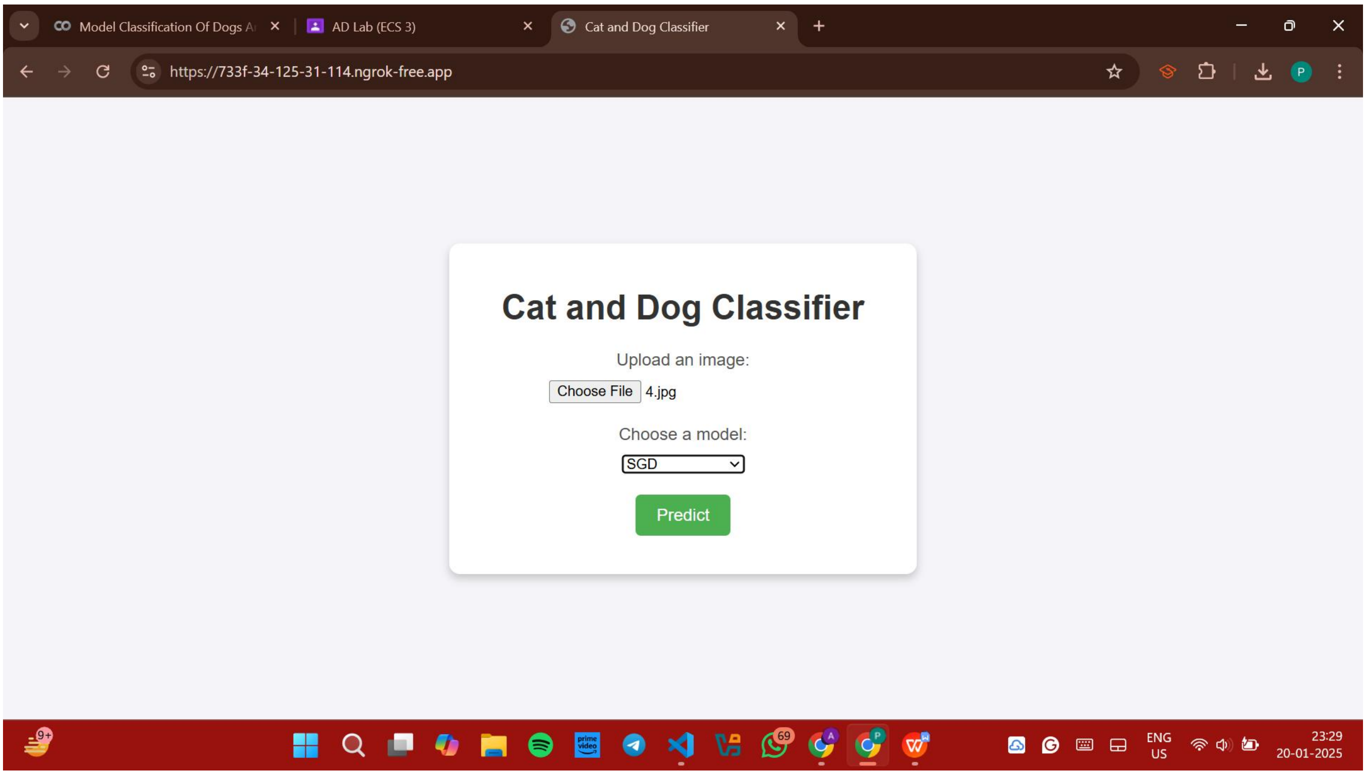

Model Classification Of Dogs A
AD Lab (ECS 3)
Prediction Result
https://733f-34-125-31-114.ngrok-free.app/predict
ENG
US
23:28
20-01-2025

Prediction Result

Model Used: rf

Prediction: Cat

Go back


Model Classification Of Dogs A
AD Lab (ECS 3)
Cat and Dog Classifier
https://733f-34-125-31-114.ngrok-free.app
ENG
US
23:29
20-01-2025

Cat and Dog Classifier

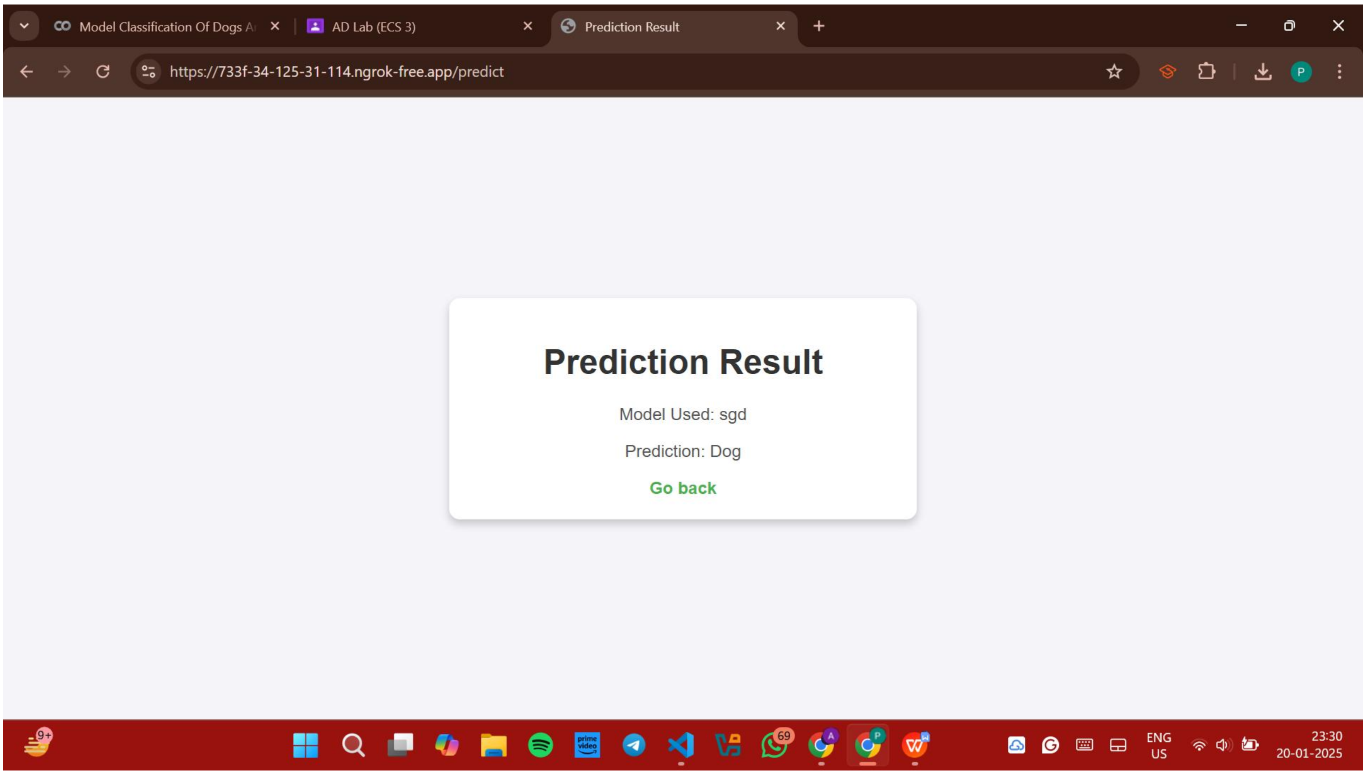
Upload an image:

Choose File 4.jpg

Choose a model:

SGD

Predict



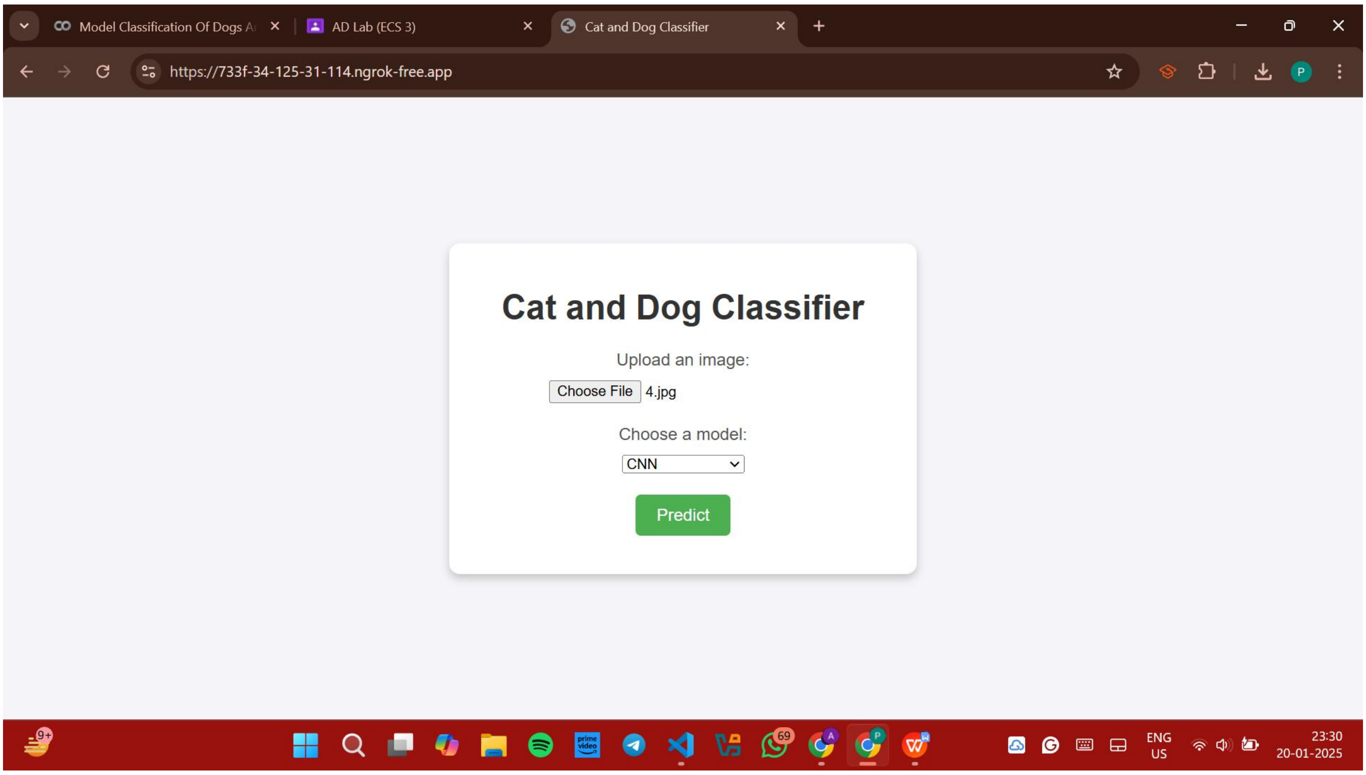
Model Classification Of Dogs A
AD Lab (ECS 3)
Prediction Result
https://733f-34-125-31-114.ngrok-free.app/predict

Prediction Result

Model Used: sgd

Prediction: Dog

Go back
ENG
US
23:30
20-01-2025



Model Classification Of Dogs A
AD Lab (ECS 3)
Cat and Dog Classifier
https://733f-34-125-31-114.ngrok-free.app

Cat and Dog Classifier

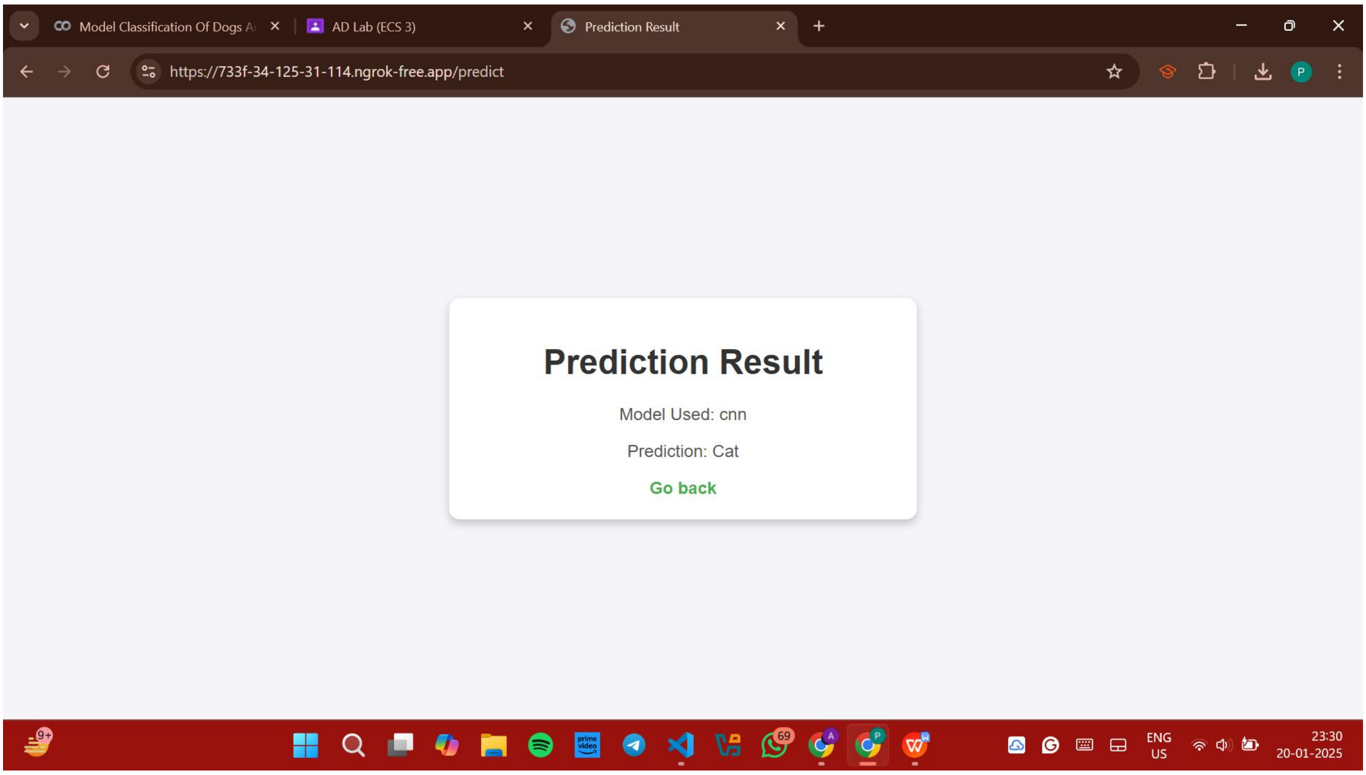
Upload an image:

Choose File 4.jpg

Choose a model:

CNN

Predict
ENG
US
23:30
20-01-2025

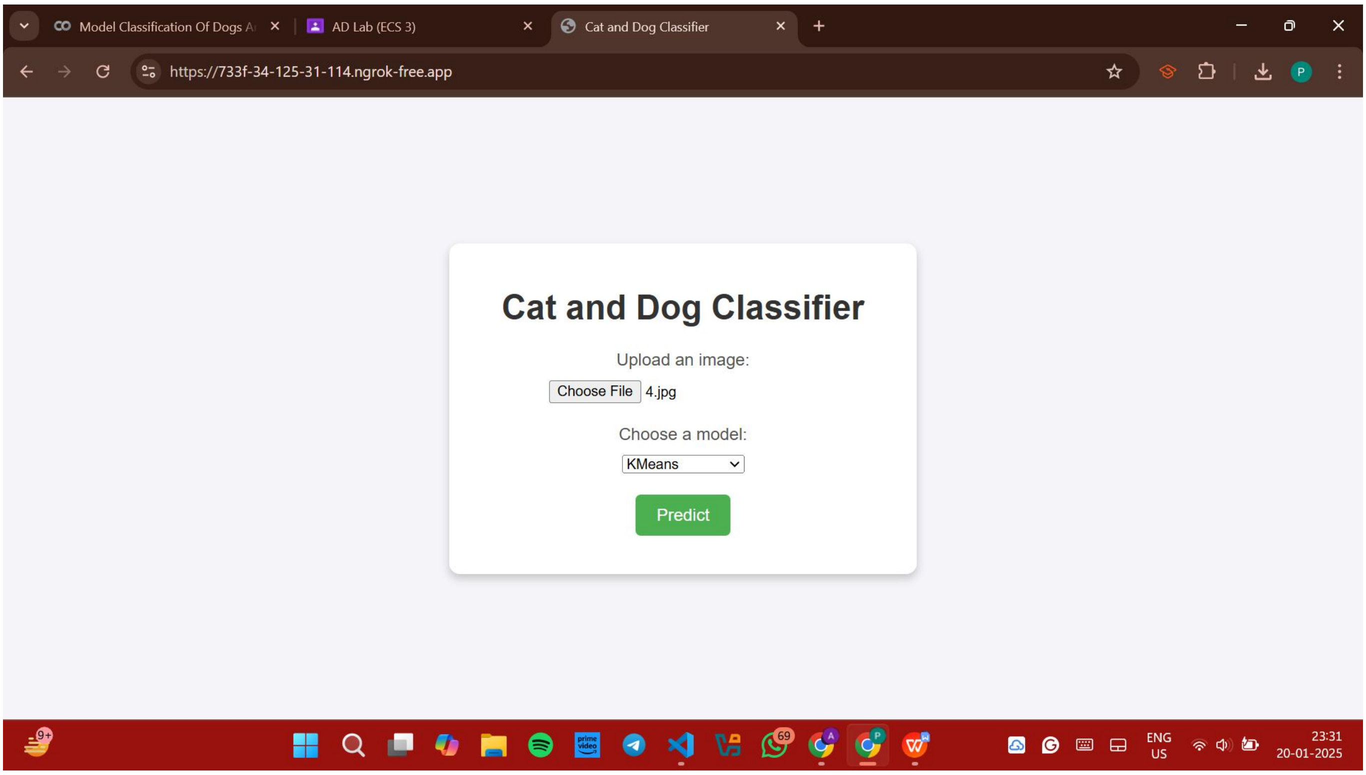

Model Classification Of Dogs A
AD Lab (ECS 3)
Prediction Result
https://733f-34-125-31-114.ngrok-free.app/predict
ENG
US
23:30
20-01-2025

Prediction Result

Model Used: cnn

Prediction: Cat

Go back


Model Classification Of Dogs A
AD Lab (ECS 3)
Cat and Dog Classifier
https://733f-34-125-31-114.ngrok-free.app
ENG
US
23:31
20-01-2025

Cat and Dog Classifier

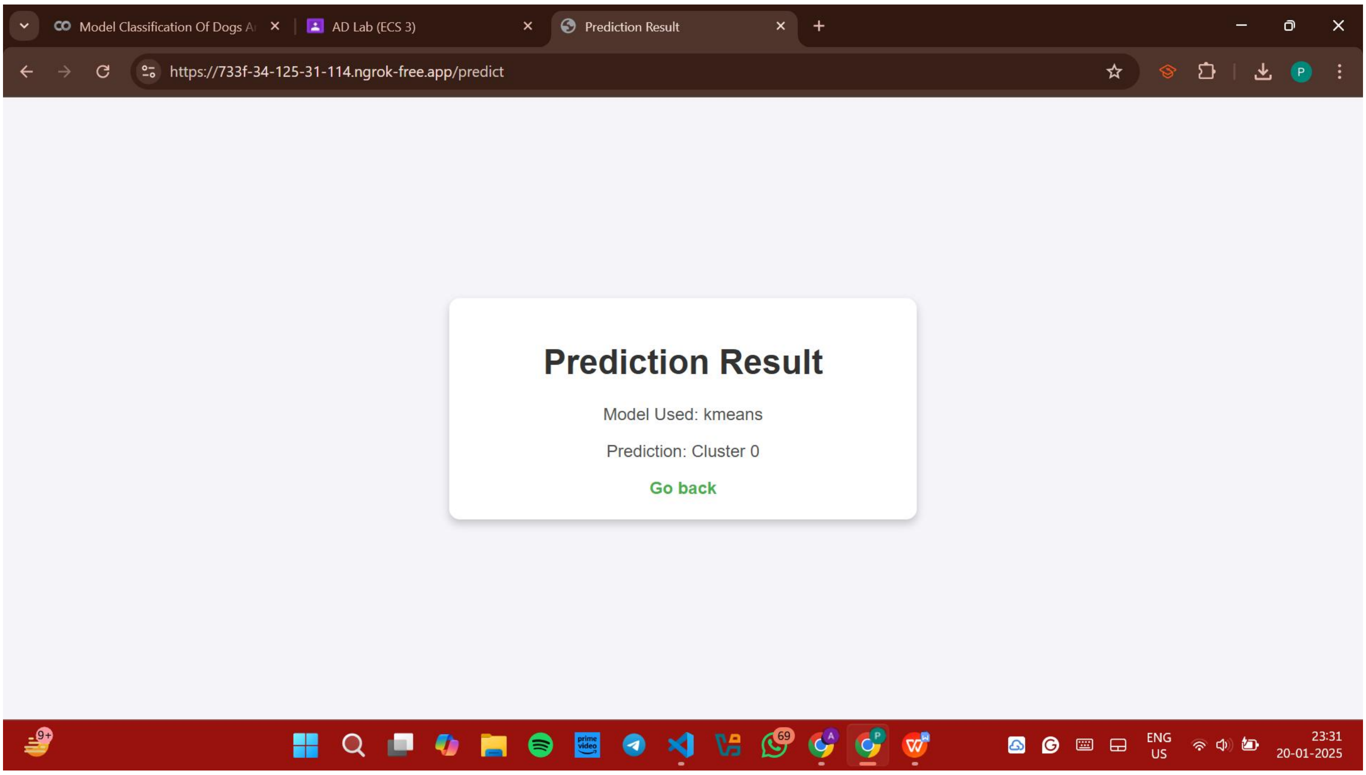
Upload an image:

Choose File 4.jpg

Choose a model:

KMeans

Predict


Model Classification Of Dogs A
AD Lab (ECS 3)
Prediction Result
https://733f-34-125-31-114.ngrok-free.app/predict
ENG
US
23:31
20-01-2025

Prediction Result

Model Used: kmeans

Prediction: Cluster 0

Go back

5. Remarks:-

In this experiment, I successfully implemented a machine learning model for classifying images of cats and dogs. The project began with collecting a labeled dataset of cat and dog images, followed by preprocessing the images using OpenCV to resize and flatten them for better model performance. I trained various machine learning models, including Support Vector Machine (SVM), Random Forest, Logistic Regression, Convolutional Neural Network (CNN), and K-means Clustering, and saved the trained models for later use. Next, I developed a Flask backend to load the trained models and handle image uploads, allowing users to interact with the models. The frontend was built using HTML and CSS, providing a simple interface for uploading images and selecting the model to use for classification.This experiment helped me gain hands-on experience in combining machine learning with web development. It enhanced my understanding of model training, data preprocessing, and building a complete web application for real-time image classification.

Signature of the Student	Signature of the Lab Coordinator
Akriti Patro	Prof. Bhargav Appasani