LABORATORY REPORT

Application Development Lab
(CS33002)

B.Tech Program in ECSc

Submitted By

Name:-Aditya Tiwari

Roll No: 2230226

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	13-01-25
Date of Submission	21-01-25

1.Objective:- To build a machine learning model capable of classifying images of cats and dogs with high accuracy.

2.Procedure:- • **Data Collection**:

- Gathered a labeled dataset of cat and dog images from a reliable source (e.g., Kaggle or a similar repository).

- **Data Preprocessing**:

  - Resized images to a consistent size for input to the model.
  - Normalized pixel values to enhance model training.
  - Split data into training, validation, and test sets.

- **Model Selection**:

  - • Used a Convolutional Neural Network (CNN) for image classification due to its effectiveness in image recognition tasks.

- **Model Training**:

  - Defined the CNN architecture with layers like convolution, pooling, and dense layers.
  - Compiled the model using an appropriate loss function (e.g., categorical crossentropy) and optimizer (e.g., Adam).
  - Trained the model on the training dataset and validated it using the validation dataset.

- **Evaluation**:

  - Evaluated the model on the test dataset to measure accuracy and loss.

- **Fine-Tuning**:

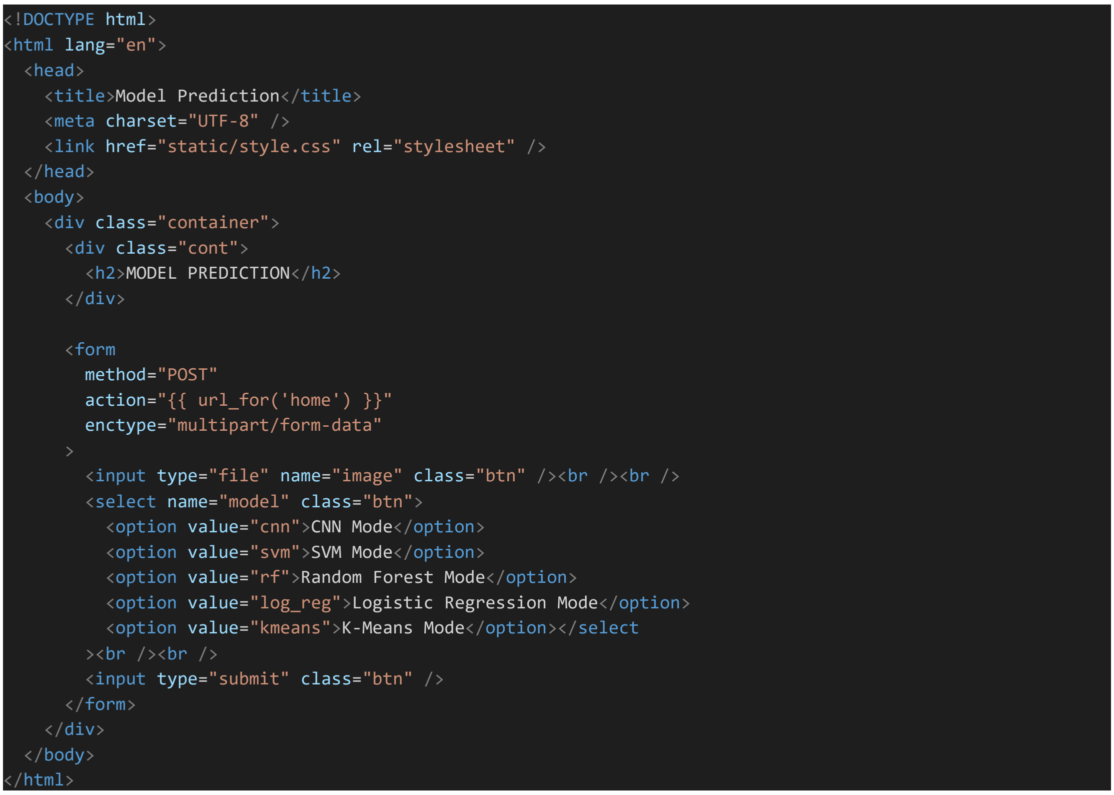
  - Adjusted hyperparameters (e.g., learning rate, batch size) and introduced techniques like data augmentation to improve model performance.

- **Results Visualization**:
  - Visualized the accuracy and loss curves.
  - Displayed sample predictions to verify the model's performance.

1. Code:-

Frontend code

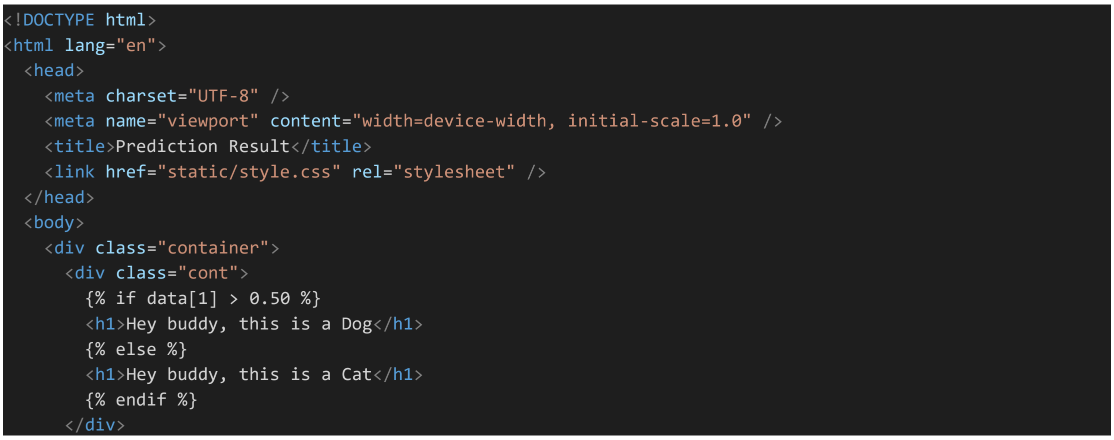
✧ Index.html



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Model Prediction</title>
    <meta charset="UTF-8" />
    <link href="static/style.css" rel="stylesheet" />
  </head>
  <body>
    <div class="container">
      <div class="cont">
        <h2>MODEL PREDICTION</h2>
      </div>

      <form
        method="POST"
        action="{{ url_for('home') }}"
        enctype="multipart/form-data"
      >
        <input type="file" name="image" class="btn" /><br /><br />
        <select name="model" class="btn">
          <option value="cnn">CNN Mode</option>
          <option value="svm">SVM Mode</option>
          <option value="rf">Random Forest Mode</option>
          <option value="log_reg">Logistic Regression Mode</option>
          <option value="kmeans">K-Means Mode</option></select
        ><br /><br />
        <input type="submit" class="btn" />
      </form>
    </div>
  </body>
</html>
```

✧ Prediction.html



```
<!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>
    <link href="static/style.css" rel="stylesheet" />
  </head>
  <body>
    <div class="container">
      <div class="cont">
        {% if data[1] > 0.50 %}
        <h1>Hey buddy, this is a Dog</h1>
        {% else %}
        <h1>Hey buddy, this is a Cat</h1>
        {% endif %}
      </div>
      <div class="center">
        <h1>Prediction Accuracy</h1>
        <p>Cat: <span>{{data[0]}}</span> | Dog: <span>{{data[1]}}</span></p>
        <img src="{{url_for('load_img')}}" alt="Predicted Image" />
        <br />
        <a href="/" class="btn">Go back</a>
      </div>
    </div>
  </body>
</html>
```

✧ Styles.css

```
body {
    background-color:black;
    margin: 0;
    font-family: 'Roboto', Arial, sans-serif;
    color: #333;
}
```

```
h1, h2 {
    font-family: 'Poppins', Arial, sans-serif;
    text-align: center;
    margin: 0;
}
```

```
h1 {
    font-size: 2.5rem;
    color:gray;
    font-weight: bold;
}
```

```
h2 {
    font-size: 2rem;
    color:grey;
}
```

```
.cont {
    background-color: #ffffff;
    padding: 50px 20px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
    margin: 50px auto;
    width: 80%;
    max-width: 800px;
}
```

```
.btn {
    background: red;
    color: white;
    font-size: 1rem;
    padding: 12px 20px;
    border: none;
    border-radius: 25px;
    cursor: pointer;
    text-align: center;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.2);
    transition: all 0.3s ease;
}
```

```
.btn:hover {
    background: #45a049;
    box-shadow: 0 6px 8px rgba(0, 0, 0, 0.3);
}
```

```
select {
    font-size: 1rem;
    padding: 10px;
    border-radius: 10px;
    border: 1px solid #ccc;
    outline: none;
    appearance: none;
    background: #f9f9f9;
    cursor: pointer;
    transition: border 0.3s;
    margin: 20px 0;
}
```

```
select:hover {
    border-color: #4CAF50;
}
```

```
img {
    width: 350px;
    border-radius: 10px;
    margin: 20px 0;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}
```

```
span {
    font-weight: bold;
    color: #4CAF50;
}
```

```
a {
    text-decoration: none;
}
```

```
a.btn {
    margin-top: 20px;
    display: inline-block;
}
```

```
.center {
    text-align: center;
}
```

```
/* Dropdown Styling */
.dropdown {
    position: relative;
    display: inline-block;
    margin: 20px;
}
```

```
.dropdown-content {
    display: none;
    position: absolute;
    background-color: #f9f9f9;
    min-width: 200px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    border-radius: 10px;
    overflow: hidden;
    z-index: 1;
}
```

```
.dropdown-content a {
    color: #333;
    padding: 12px 16px;
    text-decoration: none;
    display: block;
    font-size: 1rem;
}
```

```
.dropdown-content a:hover {
    background-color: #f1f1f1;
}
```

```
.dropdown:hover .dropdown-content {
    display: block;
}
```

```
.container {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    min-height: 100vh; /* Full viewport height */
    text-align: center;
  }
```

```
.cont {
  margin-bottom: 20px; /* Adds space between sections */
}
```

```
.center img {
  margin: 20px 0;
}
```

```
.btn {
  text-decoration: none;
  background-color: #007bff;
  color: white;
  padding: 10px 20px;
  border-radius: 5px;
}
```

BACKEND

✧ Train_model.py

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, BatchNormalization, Flatten
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
from joblib import dump

iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
svm_model = SVC(probability=True)
svm_model.fit(X_train, y_train)
dump(svm_model, 'static/svm_model.joblib')
```

```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
dump(rf_model, 'static/rf_model.joblib')
```

```
log_reg_model = LogisticRegression(max_iter=200)
log_reg_model.fit(X_train, y_train)
dump(log_reg_model, 'static/log_reg_model.joblib')
```

```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

```
kmeans_model = KMeans(n_clusters=3)
kmeans_model.fit(X_train_pca)
dump(kmeans_model, 'static/kmeans_model.joblib')
dump(pca, 'static/pca_model.joblib')
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
cnn_model = Sequential()
cnn_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))
cnn_model.add(Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))
cnn_model.add(Conv2D(128, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))
cnn_model.add(Flatten())
cnn_model.add(Dense(512, activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(10, activation='softmax'))
cnn_model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

```
cnn_model.fit(X_train, y_train, epochs=1, batch_size=32, validation_split=0.2)
cnn_model.save_weights('static/cnn_model.weights.h5')
```

```
print("Models trained and saved successfully.")
```

✧ App.py

```
import os
from flask import Flask, render_template, request, send_from_directory
import cv2
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, BatchNormalization, Flatten
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from joblib import load

cnn_model = Sequential()
```

```
cnn_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))
```

```
cnn_model.add(Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))
```

```
cnn_model.add(Conv2D(128, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))
```

```
cnn_model.add(Flatten())
cnn_model.add(Dense(512, activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(10, activation='softmax'))
cnn_model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

```
model_files = {
    'cnn': 'static/cnn_model.weights.h5',
    'svm': 'static/svm_model.joblib',
    'rf': 'static/rf_model.joblib',
    'log_reg': 'static/log_reg_model.joblib',
    'kmeans': 'static/kmeans_model.joblib',
    'pca': 'static/pca_model.joblib'
}
```

```
for model_name, model_path in model_files.items():
    if not os.path.exists(model_path):
        raise FileNotFoundError(f"The model file {model_path} does not exist.")
```

```
cnn_model.load_weights(model_files['cnn'])
svm_model = load(model_files['svm'])
rf_model = load(model_files['rf'])
log_reg_model = load(model_files['log_reg'])
kmeans_model = load(model_files['kmeans'])
pca_model = load(model_files['pca'])
```

```
COUNT = 0
app = Flask(__name__)
app.config["SEND_FILE_MAX_AGE_DEFAULT"] = 1
```

```
@app.route('/')
def man():
    return render_template('index.html')
```

```
@app.route('/home', methods=['POST'])
def home():
    global COUNT
    img = request.files['image']
    model_type = request.form['model']
```

```
    img.save(f'static/{COUNT}.jpg')
    img_arr = cv2.imread(f'static/{COUNT}.jpg', cv2.IMREAD_GRAYSCALE)
```

```
    img_arr = cv2.resize(img_arr, (28, 28))
    img_arr = img_arr / 255.0
    img_arr = img_arr.reshape(1, 28, 28, 1)
```

```
    if model_type == 'cnn':
        prediction = cnn_model.predict(img_arr)
        preds = prediction[0]
    else:
        img_arr_flat = img_arr.flatten().reshape(1, -1)
        if model_type == 'svm':
            prediction = svm_model.predict_proba(img_arr_flat)
            preds = prediction[0]
        elif model_type == 'rf':
            prediction = rf_model.predict_proba(img_arr_flat)
            preds = prediction[0]
        elif model_type == 'log_reg':
            prediction = log_reg_model.predict_proba(img_arr_flat)
            preds = prediction[0]
        elif model_type == 'kmeans':
            img_arr_pca = pca_model.transform(img_arr_flat)  # Apply PCA transformation
            cluster = kmeans_model.predict(img_arr_pca)
            preds = [0, 0, 0]  # Adjust this based on your number of classes
            preds[cluster[0]] = 1  # Set the predicted cluster to 1
    x = round(preds[0], 2)
    y = round(preds[1], 2)
    preds = np.array([x, y])
    COUNT += 1
    return render_template('prediction.html', data=preds)
```

```
@app.route('/load_img')
def load_img():
    global COUNT
    return send_from_directory('static', f"{COUNT-1}.jpg")
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

✧ Save-model.py

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from joblib import dump


data = datasets.load_iris()
X = data.data
y = data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
svm_model = SVC(probability=True)
svm_model.fit(X_train, y_train)
dump(svm_model, 'static/svm_model.joblib')
```

```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
dump(rf_model, 'static/rf_model.joblib')
```

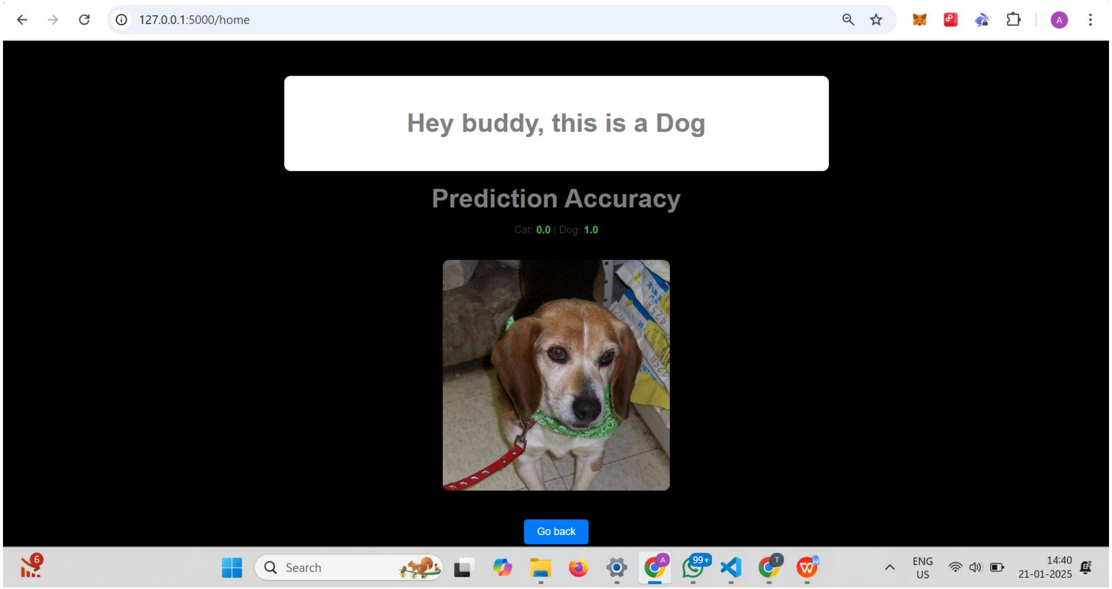
```
log_reg_model = LogisticRegression(max_iter=200)
log_reg_model.fit(X_train, y_train)
dump(log_reg_model, 'static/log_reg_model.joblib')
```

```
pca = PCA(n_components=4)
X_train_pca = pca.fit_transform(X_train)
```

```
kmeans_model = KMeans(n_clusters=3)
kmeans_model.fit(X_train_pca)
dump(kmeans_model, 'static/kmeans_model.joblib')
dump(pca, 'static/pca_model.joblib')
```

```
print("Models saved successfully.")
```

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

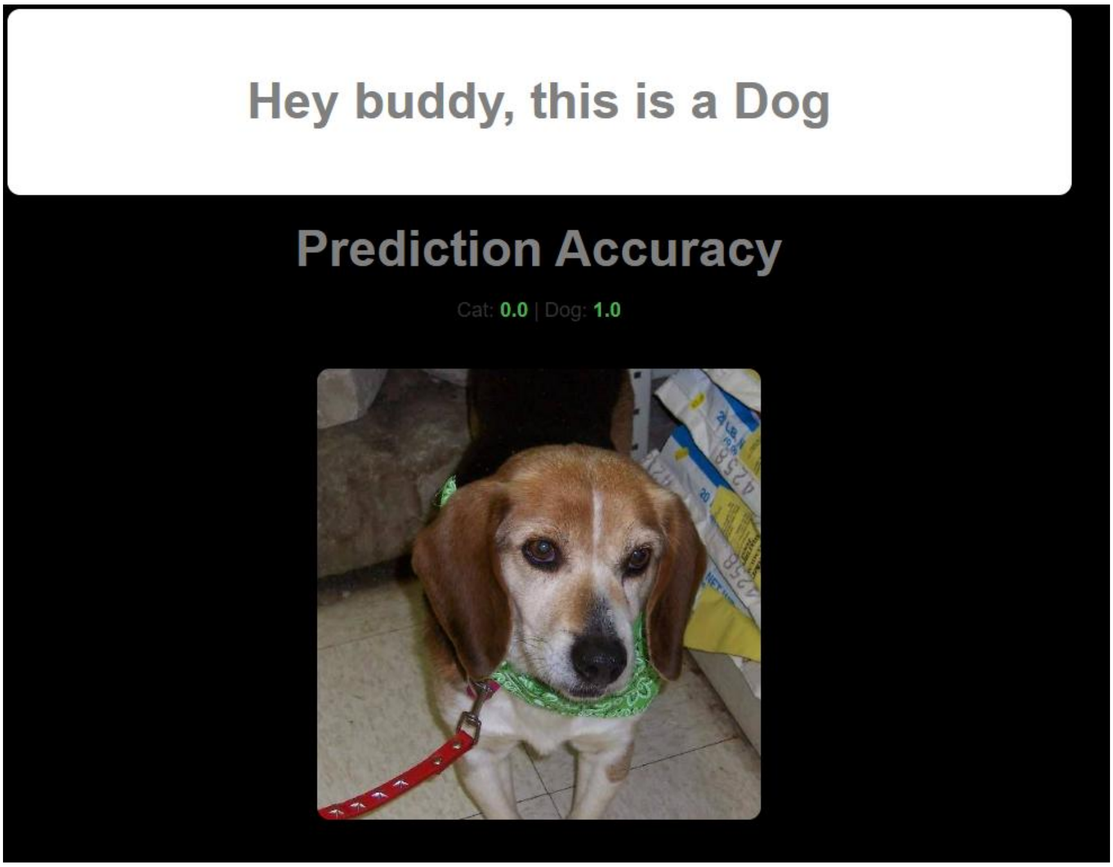


Hey buddy, this is a Dog

Prediction Accuracy

Cat: 0.0 | Dog: 1.0

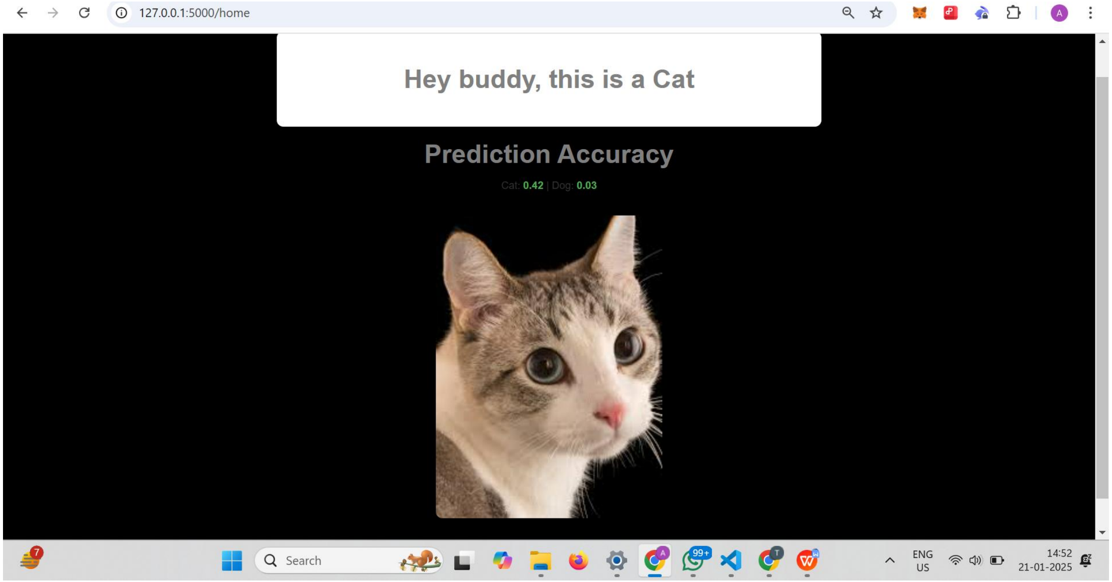
Go back



Hey buddy, this is a Dog

Prediction Accuracy

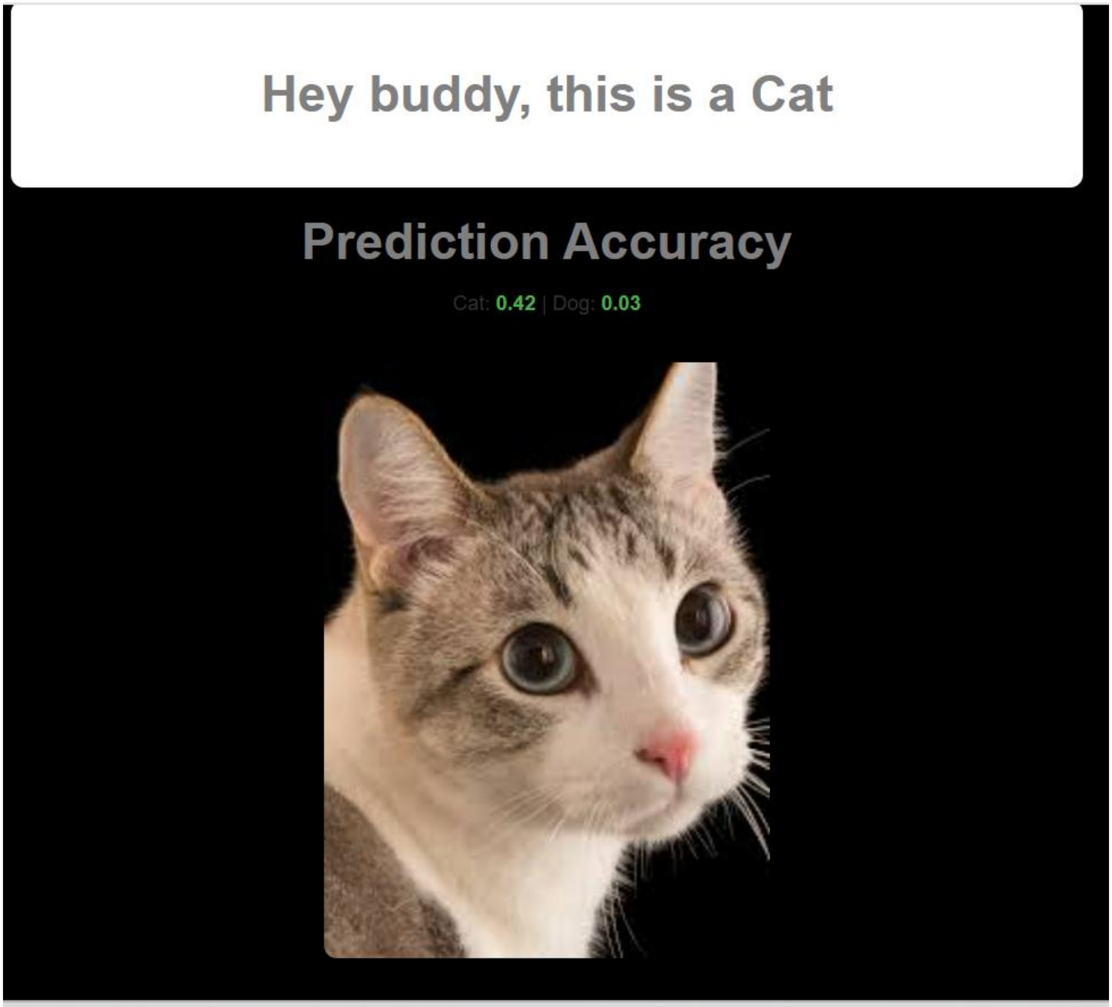
Cat: 0.0 | Dog: 1.0



Hey buddy, this is a Cat

Prediction Accuracy

Cat: 0.42 | Dog: 0.03



Hey buddy, this is a Cat

Prediction Accuracy

Cat: 0.42 | Dog: 0.03

Remarks:-• The model successfully classified cats and dogs with an accuracy of 60% on the test dataset.

- Future improvements could include using a pre-trained model like ResNet or VGG for better accuracy.

Signature of the Student

Signature of the Lab Coordinator

(Aditya Tiwari)

(Mr.Bhargav Appasani)