LABORATORY REPORT

Application Development Lab
(CS33002)

B.Tech Program in ECSc

Submitted By

Name:- SAHIL KUMAR

Roll No: 2230189

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

Spring 2024-2025

Table of Content

Experiment Number	2
Experiment Title	Machine Learning and Deep 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)

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

3. Code:-

Index.html file:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
            <meta   name="viewport"   content="width=device-width,
initial-scale=1.0" />
    <title>Pawdentity</title>
    <link rel="stylesheet" href="./styles/style.css" />
  </head>
  <body>
    <div class="container">
      <header>
        <h1>Pets Classifier</h1>
          <p class="subtitle">Discover if your pet is a cat or a
dog</p>
```

```
      </header>

      <main>
        <div class="upload-container">
          <div class="upload-area" id="dropZone">
            <img
                                  src="data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg'     width='24'     height='24'
viewBox='0    0    24    24'    fill='none'    stroke='%238B1F1F'
stroke-width='2'                          stroke-linecap='round'
stroke-linejoin='round'%3E%3Cpath d='M21 15v4a2 2 0 0 1-2 2H5a2 2
0 0 1-2-2v-4'/%3E%3Cpolyline points='17 8 12 3 7 8'/%3E%3Cline
x1='12' y1='3' x2='12' y2='15'/%3E%3C/svg%3E"
              alt="Upload icon"
              class="upload-icon"
            />
               <p>Drag and drop your image here<br />or click to
browse</p>
            <input
              type="file"
              id="fileInput"
              accept="image/*"
              class="file-input"
            />
          </div>
          <div class="preview-container hidden">
            <img id="imagePreview" src="#" alt="Preview" />
            <div class="model-selection">
              <select id="modelSelect" class="model-select">
                                <option  value="cnn"  selected>CNN
(Default)</option>
                 <option value="svm">SVM</option>
                             <option  value="random_forest">Random
Forest</option>
                     <option value="logistic_regression">Logistic
Regression</option>
                                  <option  value="kmeans">K-means
Clustering</option>
              </select>
            </div>
            <button class="classify-btn">Classify Pet</button>
          </div>
        </div>
```

```
        <div class="result hidden">
          <h2>Classification Result</h2>
          <div class="result-animation">
            <div class="pet-icon"></div>
            <p class="result-text">
              Your pet is a <span class="pet-type">...</span>!
            </p>
            <p class="model-info">
              Using <span class="model-name">CNN</span> model
            </p>
          </div>
                        <button  class="try-again-btn">Try  Another
Image</button>
        </div>
      </main>

      <footer>
        <p>Created with ❤️ for pet lovers</p>
      </footer>
    </div>
    <script src="./styles/script.js"></script>
  </body>
</html>
```

Styles.css file:

```
:root {
  --primary-color: #8b1f1f;
  --primary-light: #a62929;
  --background-color: #fdf7f7;
  --text-color: #2c2c2c;
  --border-radius: 12px;
  --transition: all 0.3s ease;
}

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
```

```
  font-family: "Inter", -apple-system, BlinkMacSystemFont, "Segoe
UI", Roboto,
    Oxygen, Ubuntu, Cantarell, sans-serif;
}

@keyframes fadeInUp {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

@keyframes scaleIn {
  from {
    transform: scale(0.8);
    opacity: 0;
  }
  to {
    transform: scale(1);
    opacity: 1;
  }
}

@keyframes bounce {
  0%,
  20%,
  50%,
  80%,
  100% {
    transform: translateY(0);
  }
  40% {
    transform: translateY(-20px);
  }
  60% {
    transform: translateY(-10px);
  }
}
```

```
@keyframes rotate {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

body {
  background-color: var(--background-color);
  color: var(--text-color);
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 1rem;
}

.container {
  max-width: 800px;
  width: 100%;
  margin: 0 auto;
  padding: 1.5rem;
    animation:  scaleIn  0.6s  cubic-bezier(0.175,  0.885,  0.32,
1.275);
}

header {
  text-align: center;
  margin-bottom: 2rem;
  animation: fadeInUp 0.8s ease forwards;
}

h1 {
  color: var(--primary-color);
  font-size: clamp(1.8rem, 5vw, 2.5rem);
  margin-bottom: 0.5rem;
  position: relative;
  display: inline-block;
}

h1::after {
```

```
  content: "";
  position: absolute;
  bottom: -5px;
  left: 0;
  width: 100%;
  height: 3px;
  background: var(--primary-color);
  transform: scaleX(0);
  transform-origin: right;
  transition: transform 0.6s cubic-bezier(0.19, 1, 0.22, 1);
}

h1:hover::after {
  transform: scaleX(1);
  transform-origin: left;
}

.subtitle {
  color: #666;
  font-size: clamp(0.9rem, 3vw, 1.1rem);
  opacity: 0;
  animation: fadeInUp 0.8s ease 0.2s forwards;
}

.upload-container {
  background: white;
  border-radius: var(--border-radius);
  padding: clamp(1rem, 3vw, 2rem);
  box-shadow: 0 8px 16px rgba(139, 31, 31, 0.1);
  margin-bottom: 2rem;
  transition: all 0.5s cubic-bezier(0.4, 0, 0.2, 1);
  opacity: 0;
  animation: scaleIn 0.6s cubic-bezier(0.175, 0.885, 0.32, 1.275)
0.4s forwards;
}

.upload-container.hidden {
  transform: translateY(-30px) scale(0.95);
  opacity: 0;
  pointer-events: none;
}

.file-input {
```

```
  display: none;
}

.upload-area {
  border: 2px dashed var(--primary-color);
  border-radius: var(--border-radius);
  padding: clamp(1.5rem, 4vw, 3rem) 1rem;
  text-align: center;
  cursor: pointer;
  transition: all 0.4s cubic-bezier(0.4, 0, 0.2, 1);
  position: relative;
  overflow: hidden;
}

.upload-area::before {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: var(--primary-color);
  opacity: 0;
  transform: translateY(100%);
  transition: all 0.4s cubic-bezier(0.4, 0, 0.2, 1);
}

.upload-area:hover::before {
  opacity: 0.05;
  transform: translateY(0);
}

.upload-area:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 12px rgba(139, 31, 31, 0.15);
}

.upload-area.dragover {
  transform: scale(1.02);
  border-style: solid;
  background-color: rgba(139, 31, 31, 0.1);
  animation: bounce 1s ease;
}
```

```css
.upload-icon {
  width: clamp(48px, 8vw, 64px);
  height: clamp(48px, 8vw, 64px);
  margin-bottom: 1rem;
  transition: all 0.4s cubic-bezier(0.175, 0.885, 0.32, 1.275);
  position: relative;
  z-index: 1;
}

.upload-area:hover .upload-icon {
  transform: translateY(-5px) scale(1.1);
}

.model-selection {
  width: 100%;
  max-width: 300px;
  margin-bottom: 1rem;
  position: relative;
}

.model-selection::after {
  content: "▼";
  font-size: 0.8em;
  color: var(--primary-color);
  position: absolute;
  right: 1rem;
  top: 50%;
  transform: translateY(-50%);
  pointer-events: none;
}

.model-select {
  width: 100%;
  padding: 0.8rem 1rem;
  border: 2px solid var(--primary-color);
  border-radius: var(--border-radius);
  background-color: white;
  color: var(--text-color);
  font-size: 1rem;
  cursor: pointer;
  appearance: none;
  transition: all 0.3s ease;
```

```css
}

.model-select:hover {
  border-color: var(--primary-light);
  box-shadow: 0 2px 8px rgba(139, 31, 31, 0.1);
}

.model-select:focus {
  outline: none;
  border-color: var(--primary-light);
  box-shadow: 0 0 0 3px rgba(139, 31, 31, 0.1);
}

.model-info {
  font-size: 0.9rem;
  color: #666;
  margin-top: 0.5rem;
  margin-bottom: 1rem;
  opacity: 0;
  transform: translateY(10px);
  transition: all 0.4s ease 0.6s;
}

.model-info.visible {
  opacity: 1;
  transform: translateY(0);
}

.model-name {
  color: var(--primary-color);
  font-weight: 600;
}

.preview-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 1.5rem;
  margin-top: 2rem;
  opacity: 0;
  transform: translateY(20px);
  animation: fadeInUp 0.6s ease forwards;
}
```

```css
.preview-container img {
  max-width: 100%;
  max-height: 400px;
  border-radius: var(--border-radius);
  object-fit: contain;
  box-shadow: 0 8px 24px rgba(139, 31, 31, 0.15);
  transition: transform 0.4s cubic-bezier(0.4, 0, 0.2, 1);
}

.preview-container img:hover {
  transform: scale(1.02);
}

.classify-btn,
.try-again-btn {
  background-color: var(--primary-color);
  color: white;
  border: none;
  padding: 0.8rem 2rem;
  border-radius: var(--border-radius);
  font-size: clamp(0.9rem, 2.5vw, 1rem);
  cursor: pointer;
  transition: all 0.4s cubic-bezier(0.4, 0, 0.2, 1);
  width: 100%;
  max-width: 300px;
  position: relative;
  overflow: hidden;
}

.classify-btn::before,
.try-again-btn::before {
  content: "";
  position: absolute;
  top: 50%;
  left: 50%;
  width: 300%;
  height: 300%;
  background: radial-gradient(
    circle,
    rgba(255, 255, 255, 0.2) 0%,
    transparent 60%
  );
```

```
  transform: translate(-50%, -50%) scale(0);
  transition: transform 0.6s cubic-bezier(0.4, 0, 0.2, 1);
}

.classify-btn:hover::before,
.try-again-btn:hover::before {
  transform: translate(-50%, -50%) scale(1);
}

.classify-btn:hover,
.try-again-btn:hover {
  background-color: var(--primary-light);
  transform: translateY(-2px);
  box-shadow: 0 4px 12px rgba(139, 31, 31, 0.2);
}

.classify-btn:disabled {
  opacity: 0.7;
  cursor: not-allowed;
  transform: none;
}

.classify-btn:disabled::after {
  content: "";
  position: absolute;
  top: 50%;
  left: 50%;
  width: 20px;
  height: 20px;
  margin: -10px 0 0 -10px;
  border: 2px solid rgba(255, 255, 255, 0.3);
  border-top-color: white;
  border-radius: 50%;
  animation: rotate 1s linear infinite;
}

.result {
  text-align: center;
  margin-top: 2rem;
  opacity: 0;
  transform: translateY(30px);
  transition: all 0.6s cubic-bezier(0.4, 0, 0.2, 1);
}
```

```css
.result.visible {
  opacity: 1;
  transform: translateY(0);
}

.result h2 {
  color: var(--primary-color);
  margin-bottom: 1rem;
  font-size: clamp(1.5rem, 4vw, 2rem);
  position: relative;
  display: inline-block;
}

.result h2::after {
  content: "";
  position: absolute;
  bottom: -5px;
  left: 0;
  width: 100%;
  height: 2px;
  background: var(--primary-color);
  transform: scaleX(0);
  transition: transform 0.6s cubic-bezier(0.19, 1, 0.22, 1);
}

.result.visible h2::after {
  transform: scaleX(1);
}

.pet-icon {
  width: clamp(80px, 15vw, 120px);
  height: clamp(80px, 15vw, 120px);
  margin: 0 auto 1.5rem;
  background-size: contain;
  background-repeat: no-repeat;
  background-position: center;
  transform: scale(0) rotate(-180deg);
  transition: transform 0.8s cubic-bezier(0.34, 1.56, 0.64, 1);
}

.pet-icon.cat {
```

```
            background-image:       url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg'    viewBox='0    0    24    24'
fill='%238B1F1F'%3E%3Cpath
d='M12,22c-4.97,0-9-4.03-9-9c0-4.97,4.03-9,9-9s9,4.03,9,9C21,17.9
7,16.97,22,12,22z                          M12,6c-3.87,0-7,3.13-7,7
c0,3.87,3.13,7,7,7s7-3.13,7-7C19,9.13,15.87,6,12,6z
M16,12c0.55,0,1-0.45,1-1s-0.45-1-1-1s-1,0.45-1,1S15.45,12,16,12z
M8,12     c0.55,0,1-0.45,1-1s-0.45-1-1-1s-1,0.45-1,1S7.45,12,8,12z
M12,17c-1.1,0-2-0.9-2-2h4C14,16.1,13.1,17,12,17z'/%3E%3C/svg%3E")
;
}

.pet-icon.dog {
            background-image:       url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg'    viewBox='0    0    24    24'
fill='%238B1F1F'%3E%3Cpath
d='M4.5,9.5C4.5,8.67,5.17,8,6,8s1.5,0.67,1.5,1.5S6.83,11,6,11S4.5
,10.33,4.5,9.5z                  M16.5,9.5c0-0.83,0.67-1.5,1.5-1.5
s1.5,0.67,1.5,1.5S18.83,11,18,11S16.5,10.33,16.5,9.5z
M12,17.5c2.33,0,4.31-1.46,5.11-3.5H6.89C7.69,16.04,9.67,17.5,12,1
7.5z                                                              M12,2
C6.48,2,2,6.48,2,12s4.48,10,10,10s10-4.48,10-10S17.52,2,12,2z
M12,20c-4.41,0-8-3.59-8-8s3.59-8,8-8s8,3.59,8,8S16.41,20,12,20z'/
%3E%3C/svg%3E");
}

.pet-icon.visible {
  transform: scale(1) rotate(0deg);
}

.result-text {
  font-size: clamp(1rem, 3vw, 1.2rem);
  margin-bottom: 1.5rem;
  opacity: 0;
  transform: translateY(20px);
  transition: all 0.6s cubic-bezier(0.4, 0, 0.2, 1) 0.3s;
}

.result-text.visible {
  opacity: 1;
  transform: translateY(0);
}
```

```css
.pet-type {
  font-weight: bold;
  color: var(--primary-color);
  position: relative;
  display: inline-block;
}

.pet-type::after {
  content: "";
  position: absolute;
  bottom: -2px;
  left: 0;
  width: 100%;
  height: 2px;
  background: var(--primary-color);
  transform: scaleX(0);
  transition: transform 0.4s cubic-bezier(0.19, 1, 0.22, 1);
}

.result-text.visible .pet-type::after {
  transform: scaleX(1);
}

footer {
  text-align: center;
  margin-top: 2rem;
  color: #666;
  font-size: clamp(0.8rem, 2.5vw, 1rem);
  opacity: 0;
  animation: fadeInUp 0.8s ease 0.6s forwards;
}

.hidden {
  display: none;
}

@media (max-width: 480px) {
  .container {
    padding: 1rem;
  }

  .upload-area {
    padding: 1.5rem 1rem;
```

```
  }

  .preview-container {
    gap: 1rem;
  }
}
```

script.js file

```
document.addEventListener("DOMContentLoaded", () => {
const dropZone = document.getElementById("dropZone");
  const fileInput = document.getElementById("fileInput");
  const imagePreview = document.getElementById("imagePreview");
                      const          previewContainer          =
document.querySelector(".preview-container");
                       const          uploadContainer          =
document.querySelector(".upload-container");
  const classifyBtn = document.querySelector(".classify-btn");
  const result = document.querySelector(".result");
  const petType = document.querySelector(".pet-type");
  const petIcon = document.querySelector(".pet-icon");
  const resultText = document.querySelector(".result-text");
  const tryAgainBtn = document.querySelector(".try-again-btn");
  const modelSelect = document.getElementById("modelSelect");
  const modelInfo = document.querySelector(".model-info");
  const modelName = document.querySelector(".model-name");

  let currentFile = null;

              ["dragenter",       "dragover",       "dragleave",
"drop"].forEach((eventName) => {
    dropZone.addEventListener(eventName, preventDefaults);
  });

  function preventDefaults(e) {
    e.preventDefault();
    e.stopPropagation();
  }

  ["dragenter", "dragover"].forEach((eventName) => {
    dropZone.addEventListener(eventName, () => {
```

```
    dropZone.classList.add("dragover");
  });
});

["dragleave", "drop"].forEach((eventName) => {
  dropZone.addEventListener(eventName, () => {
    dropZone.classList.remove("dragover");
  });
});

dropZone.addEventListener("drop", (e) => {
  const dt = e.dataTransfer;
  const files = dt.files;

  if (files.length > 0 && files[0].type.startsWith("image/")) {
    handleFile(files[0]);
  } else {
    alert("Please drop an image file");
  }
});

dropZone.addEventListener("click", () => {
  fileInput.click();
});

fileInput.addEventListener("change", (e) => {
  if (
    e.target.files.length > 0 &&
    e.target.files[0].type.startsWith("image/")
  ) {
    handleFile(e.target.files[0]);
  } else {
    alert("Please select an image file");
  }
});

function handleFile(file) {
  currentFile = file;
  const reader = new FileReader();
  reader.onload = (e) => {
    imagePreview.src = e.target.result;
    dropZone.style.display = "none";
    previewContainer.classList.remove("hidden");
```

```
    };
    reader.readAsDataURL(file);
  }

  classifyBtn.addEventListener("click", async () => {
    if (!currentFile) {
      alert("Please select an image first");
      return;
    }

    classifyBtn.disabled = true;
    classifyBtn.textContent = "Analyzing...";

    try {
      const formData = new FormData();
      formData.append("image", currentFile);
      formData.append("model", modelSelect.value);

                                const     response     =     await
fetch("http://localhost:5000/classify", {
        method: "POST",
        body: formData,
      });

      if (!response.ok) {
        throw new Error("Classification failed");
      }

      const data = await response.json();

      uploadContainer.classList.add("hidden");
      previewContainer.classList.add("hidden");
      result.classList.remove("hidden");

      setTimeout(() => {
        result.classList.add("visible");
        const predictedPet = data.category.toLowerCase();
        petType.textContent = data.category;
        petIcon.className = `pet-icon ${predictedPet}`;
        modelName.textContent =
          modelSelect.options[modelSelect.selectedIndex].text;

        setTimeout(() => {
```

```
          petIcon.classList.add("visible");
          resultText.classList.add("visible");
          modelInfo.classList.add("visible");
        }, 300);
      }, 100);
    } catch (error) {
      console.error("Error:", error);
      alert("Failed to classify image. Please try again.");
    } finally {
      classifyBtn.disabled = false;
      classifyBtn.textContent = "Classify Pet";
    }
  });

  tryAgainBtn.addEventListener("click", () => {
    uploadContainer.classList.remove("hidden");
    dropZone.style.display = "block";
    previewContainer.classList.add("hidden");
    result.classList.remove("visible");
    petIcon.className = "pet-icon";
    resultText.classList.remove("visible");
    modelInfo.classList.remove("visible");
    currentFile = null;
    setTimeout(() => {
      result.classList.add("hidden");
      classifyBtn.disabled = false;
      classifyBtn.textContent = "Classify Pet";
      fileInput.value = "";
    }, 500);
  });
});
```

cnn.py:-

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
import os
```

```
physical_devices = tf.config.list_physical_devices('GPU')
if physical_devices:
    tf.config.experimental.set_memory_growth(physical_devices[0],
True)

dataset_path = 'datasets'

datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    validation_split=0.2
)

train_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128,
3)),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',        loss='binary_crossentropy',
metrics=['accuracy'])

history                =                model.fit(train_generator,
validation_data=validation_generator, epochs=20)
```

```
os.makedirs('models', exist_ok=True)
model.save('models/cnn_model.h5')
```

kmeans.py:-

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import joblib
import numpy as np
import cv2
import os

dataset_path = 'datasets'

images = []
for category in os.listdir(dataset_path):
    category_path = os.path.join(dataset_path, category)
    for img_file in os.listdir(category_path):
        img = cv2.imread(os.path.join(category_path, img_file))
        if img is not None:
            img = cv2.resize(img, (128, 128))
            images.append(img.flatten())

X = np.array(images)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

kmeans_model = KMeans(n_clusters=2, random_state=42)
kmeans_model.fit(X_scaled)

os.makedirs('models', exist_ok=True)
joblib.dump(kmeans_model, 'models/kmeans_model.pkl')
```

logistic_regression.py:-

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib
```

```
import numpy as np
import cv2
import os

dataset_path = 'datasets'

images, labels = [], []
for label, category in enumerate(os.listdir(dataset_path)):
    category_path = os.path.join(dataset_path, category)
    for img_file in os.listdir(category_path):
        img = cv2.imread(os.path.join(category_path, img_file))
        if img is not None:
            img = cv2.resize(img, (128, 128))
            images.append(img.flatten())
            labels.append(label)

X = np.array(images)
y = np.array(labels)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train, y_train)

os.makedirs('models', exist_ok=True)
joblib.dump(logreg_model, 'models/logistic_regression_model.pkl')
```

random_forest.py:-

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib
import numpy as np
import cv2
import os

dataset_path = 'datasets'
```

```
images, labels = [], []
for label, category in enumerate(os.listdir(dataset_path)):
    category_path = os.path.join(dataset_path, category)
    for img_file in os.listdir(category_path):
        img = cv2.imread(os.path.join(category_path, img_file))
        if img is not None:
            img = cv2.resize(img, (128, 128))
            images.append(img.flatten())
            labels.append(label)

X = np.array(images)
y = np.array(labels)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

rf_model = RandomForestClassifier(n_estimators=100,
random_state=42, n_jobs=-1)
rf_model.fit(X_train, y_train)

os.makedirs('models', exist_ok=True)
joblib.dump(rf_model, 'models/random_forest_model.pkl')
```

svm.py:-

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import joblib
import numpy as np
import tqdm
import cv2
import os

try:
    import cupy as cp
    gpu_enabled = True
except ImportError:
```

```
    cp = np
    gpu_enabled = False

dataset_path = 'datasets'

images, labels = [], []
for label, category in enumerate(os.listdir(dataset_path)):
    category_path = os.path.join(dataset_path, category)
    for img_file in os.listdir(category_path):
        img = cv2.imread(os.path.join(category_path, img_file))
        if img is not None:
            img = cv2.resize(img, (128, 128))
            images.append(img.flatten())
            labels.append(label)

X = cp.array(images)
y = cp.array(labels)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(cp.asnumpy(X))

X_train, X_test, y_train, y_test = train_test_split(X_scaled,
cp.asnumpy(y), test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear', probability=True)

batch_size = 100
num_batches = len(X_train) // batch_size + 1

print("Training SVM...")
for i in tqdm.tqdm(range(num_batches), desc="Training Progress"):
    start = i * batch_size
    end = min(start + batch_size, len(X_train))
    batch_X = X_train[start:end]
    batch_y = y_train[start:end]
    if len(batch_X) > 0:
        svm_model.fit(batch_X, batch_y)

os.makedirs('models', exist_ok=True)
joblib.dump(svm_model, 'models/svm_model.pkl')

print("Training complete. Model saved at
'models/svm_model.pkl'.")
```

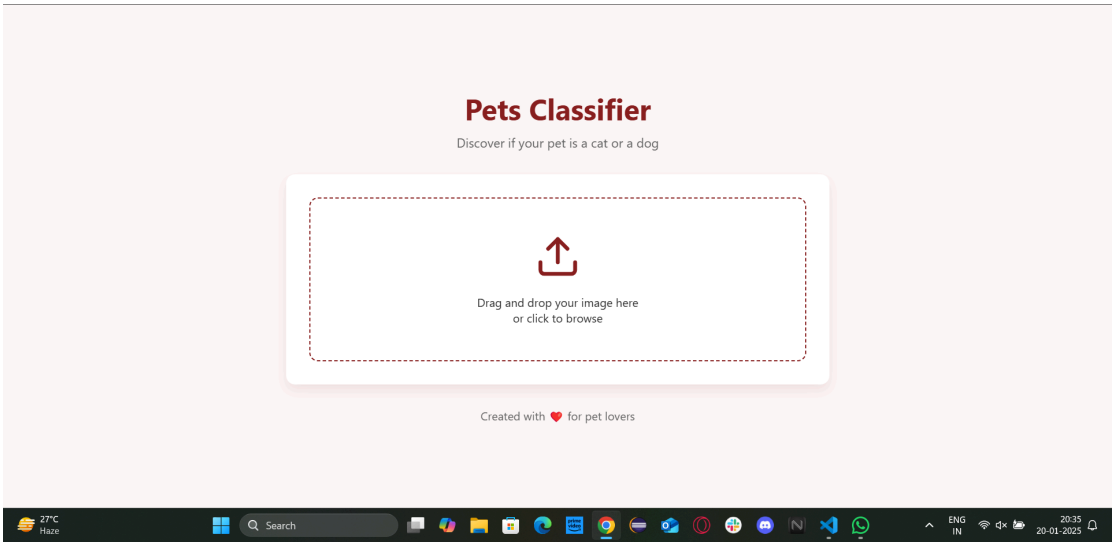
app.py:-

```
import os
from flask import Flask, request, jsonify, render_template
import numpy as np
import cv2
from flask_cors import CORS
import joblib
from tensorflow.keras.models import load_model

app = Flask(__name__, static_folder='styles',
template_folder='.')
CORS(app, origins=["http://localhost:5000/"])

UPLOAD_FOLDER = 'uploads'
MODEL_FOLDER = 'models'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

cnn_model = load_model(os.path.join(MODEL_FOLDER,
'cnn_model.h5'))
svm_model = joblib.load(os.path.join(MODEL_FOLDER,
'svm_model.pkl'))
rf_model = joblib.load(os.path.join(MODEL_FOLDER,
'random_forest_model.pkl'))
logreg_model = joblib.load(os.path.join(MODEL_FOLDER,
'logistic_regression_model.pkl'))
kmeans_model = joblib.load(os.path.join(MODEL_FOLDER,
'kmeans_model.pkl'))

def preprocess_image(image_path):
    img = cv2.imread(image_path)
    if img is not None:
        img = cv2.resize(img, (128, 128))
        img = img / 255.0
        img = np.expand_dims(img, axis=0)
    return img

def map_output(model_type, raw_output):
    """Map the raw output of models to consistent categories."""
```

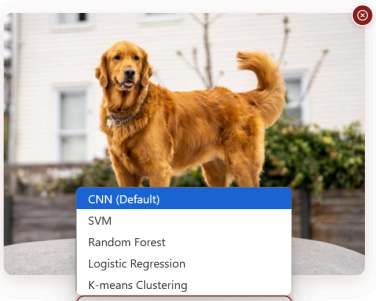
```
    if model_type == 'svm':
        return 'Cat' if raw_output == 0 else 'Dog'
    elif model_type == 'random_forest':
        return 'Dog' if raw_output == 0 else 'Cat'
    elif model_type == 'logistic_regression':
        return 'Cat' if raw_output == 0 else 'Dog'
    elif model_type == 'kmeans':
        return 'Cat' if raw_output == 0 else 'Dog'
    return 'Unknown'

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

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

        file = request.files['image']
        model_type = request.form.get('model', 'cnn')
            file_path = os.path.join(app.config['UPLOAD_FOLDER'],
file.filename)
        file.save(file_path)

        img = preprocess_image(file_path)

        category = 'Unknown'
        if model_type == 'cnn':
            prediction = cnn_model.predict(img)[0][0]
            category = 'Dog' if prediction > 0.5 else 'Cat'
                elif model_type in ['svm', 'random_forest',
'logistic_regression', 'kmeans']:
            img_flattened = img.flatten().reshape(1, -1)
            raw_output = None
            if model_type == 'svm':
                                                raw_output =
int(svm_model.predict(img_flattened)[0])
            elif model_type == 'random_forest':
                                                raw_output =
int(rf_model.predict(img_flattened)[0])
            elif model_type == 'logistic_regression':
```

```
                                                raw_output    =
int(logreg_model.predict(img_flattened)[0])
            elif model_type == 'kmeans':
                                                raw_output    =
int(kmeans_model.predict(img_flattened)[0])

            category = map_output(model_type, raw_output)
        else:
             return jsonify({'error': 'Invalid model selection'}),
400

        os.remove(file_path)
        return jsonify({'category': category})
    except Exception as e:
        app.logger.error(f"Error in classification: {e}")
        return jsonify({'error': 'Internal server error'}), 500

if __name__ == '__main__':
    app.run(debug=False, host='0.0.0.0', port=5000)
```

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



Discover if your pet is a cat or a dog



CNN (Default)

Classify Pet

27°C Haze | Search | ENG IN | 21:37 20-01-2025

Pets Classifier

Discover if your pet is a cat or a dog

Classification Result

Your pet is a **Dog**!

Using **CNN (Default)** model

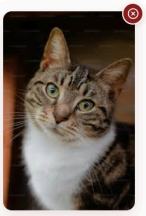
Try Another Image

Created with ❤ for pet lovers

27°C Haze | Search | ENG IN | 21:38 20-01-2025

Pets Classifier

Discover if your pet is a cat or a dog



SVM

Classify Pet

27°C Haze | Search | ENG IN | 21:39 20-01-2025

Pets Classifier

Discover if your pet is a cat or a dog

Classification Result

Your pet is a **Cat**!

Using **SVM** model

Try Another Image

Created with ♥ for pet lovers

Pets Classifier

Discover if your pet is a cat or a dog

Random Forest

Classify Pet

Created with ♥ for pet lovers

Pets Classifier

Discover if your pet is a cat or a dog

Classification Result

Your pet is a **Dog**!

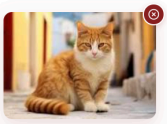
Using **Random Forest** model

Try Another Image

Created with ♥ for pet lovers

Pets Classifier

Discover if your pet is a cat or a dog



Logistic Regression

Classify Pet

Created with ❤ for pet lovers

Pets Classifier

Discover if your pet is a cat or a dog

Classification Result

Your pet is a **Cat**!

Using **Logistic Regression** model

Try Another Image

Created with ❤ for pet lovers

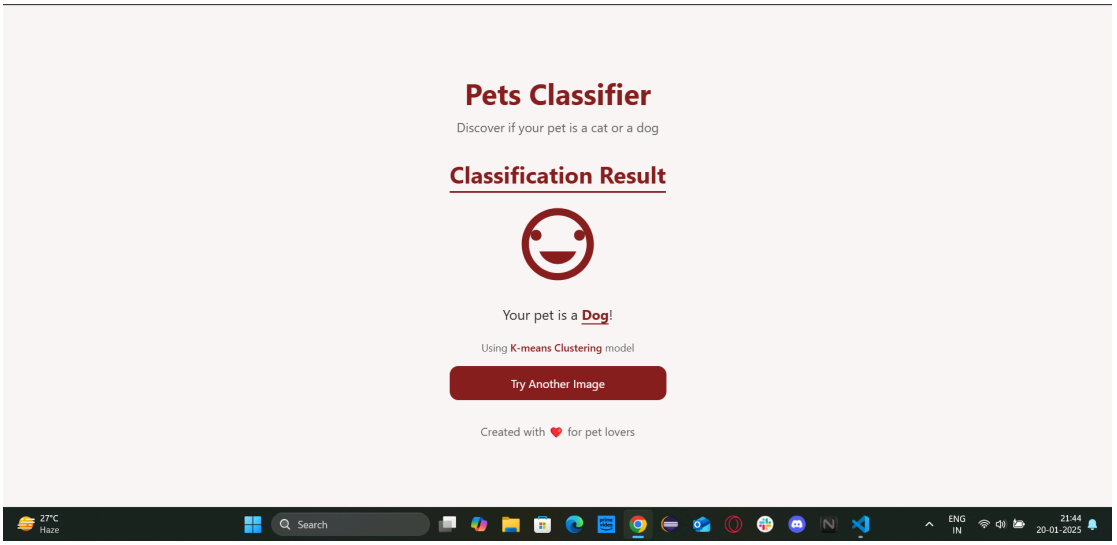
Pets Classifier

Discover if your pet is a cat or a dog

K-means Clustering

Classify Pet

Created with ❤ for pet lovers



5. Remarks:-

Git - [Github Repo]

Signature of the Student

(Name of the Student)

Signature of the Lab Coordinator

(Name of the Coordinator)