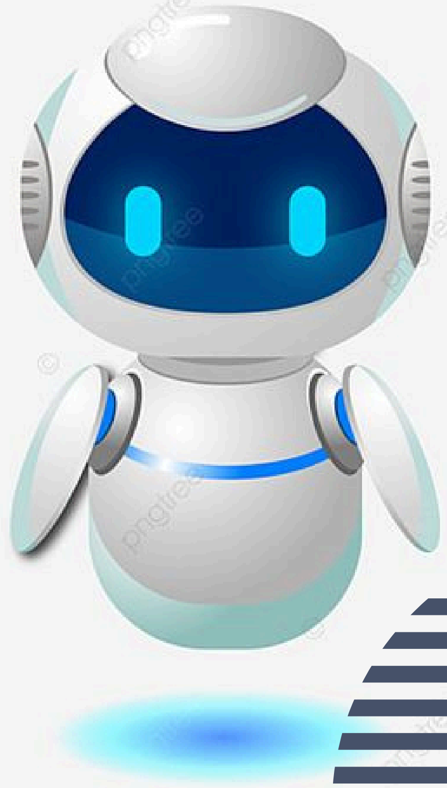


# Project AI

# supervized ML

Supervised learning is a type of machine learning where the computer is taught using examples. These examples have both the input (what you know) and the output (what you want to predict). The goal is for the computer to learn how to predict the output based on the input.

# process :

1. Data Collection: Gather a labeled dataset, where each example consists of an input-output pair.
2. Data Preprocessing: Clean the data, handle missing values, normalize or scale features, and possibly perform feature engineering.
3. Train-Test Split: Split the dataset into training and testing sets to evaluate the model's performance.
4. Model Selection: Choose a suitable algorithm for the problem at hand (e.g., linear regression for regression tasks, decision trees for classification tasks).
5. Training: Use the training data to train the model, adjusting the model’s parameters to minimize the error between predicted and actual outputs.
6. Evaluation: Assess the model's performance on the test set using appropriate metrics (e.g., accuracy, precision, recall for classification; mean squared error, R-squared for regression).
7. Hyperparameter Tuning: Optimize the model’s hyperparameters to improve performance.
8. Prediction: Use the trained model to make predictions on new, unseen data.

# Algorithm used in this project :

for classifying images of cats and dogs, we extract features from the images and then use traditional machine learning algorithms such as Decision tree and k-Nearest Neighbors (k-NN). here is a steps for what we do :.

## 1- import libraries:

```
import os
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
```

**Your paragraph text**

## 2- Load and Preprocess Images:

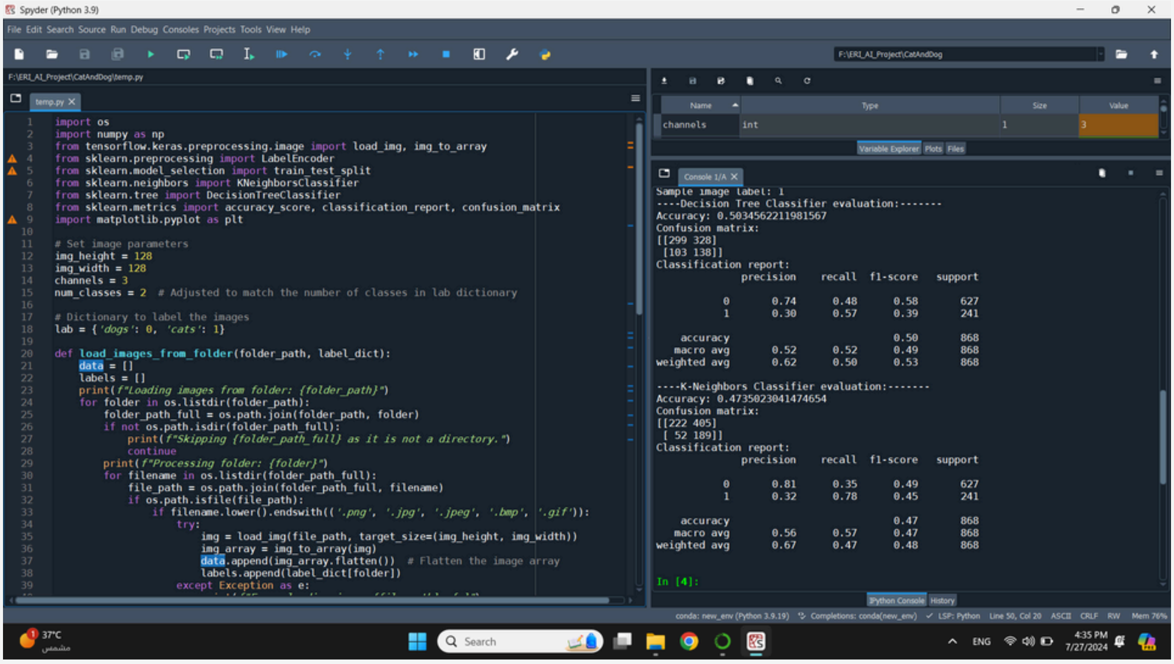
```
img_height = 128
img_width = 128
channels = 3
num_classes = 2  # Adjusted to match the number of classes in lab dictionary

# Dictionary to label the images
lab = {'dogs': 0, 'cats': 1}

def load_images_from_folder(folder_path, label_dict):
    data = []
    labels = []
    print(f"Loading images from folder: {folder_path}")
    for folder in os.listdir(folder_path):
        folder_path_full = os.path.join(folder_path, folder)
        if not os.path.isdir(folder_path_full):
            print(f"Skipping {folder_path_full} as it is not a directory.")
            continue
        print(f"Processing folder: {folder}")
        for filename in os.listdir(folder_path_full):
            file_path = os.path.join(folder_path_full, filename)
            if os.path.isfile(file_path):
                if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):
                    try:
                        img = load_img(file_path, target_size=(img_height, img_width))
                        img_array = img_to_array(img)
                        data.append(img_array.flatten())  # Flatten the image array
                        labels.append(label_dict[folder])
                    except Exception as e:
```

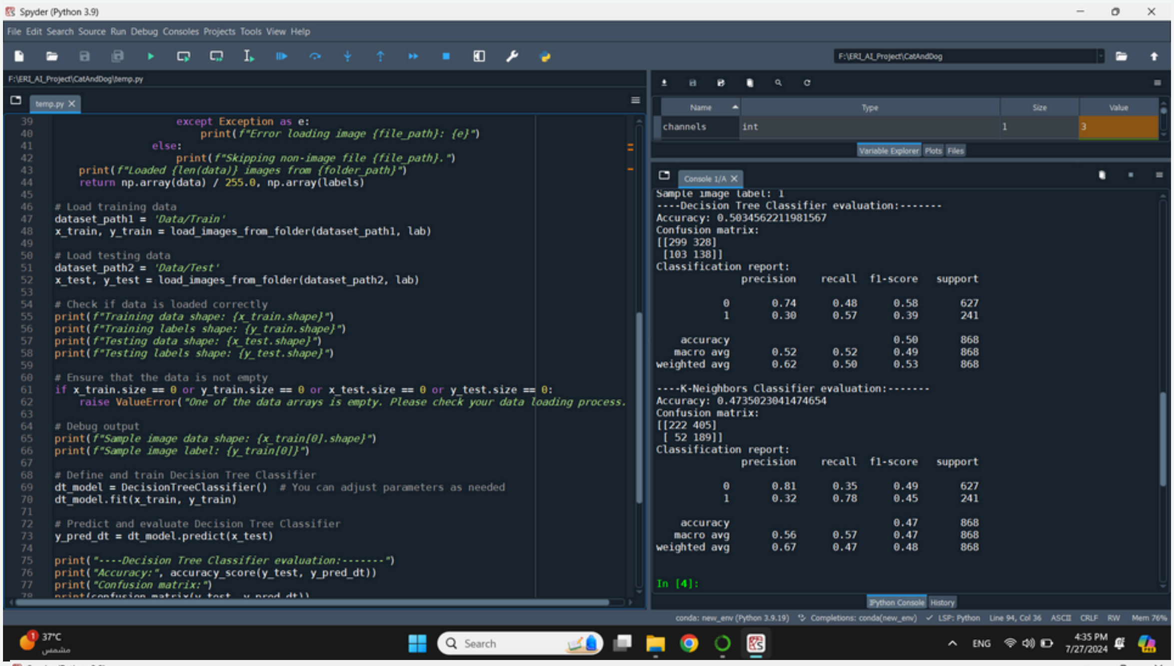
# 3-Train decision tree Model and k-nn model and evaluate them :

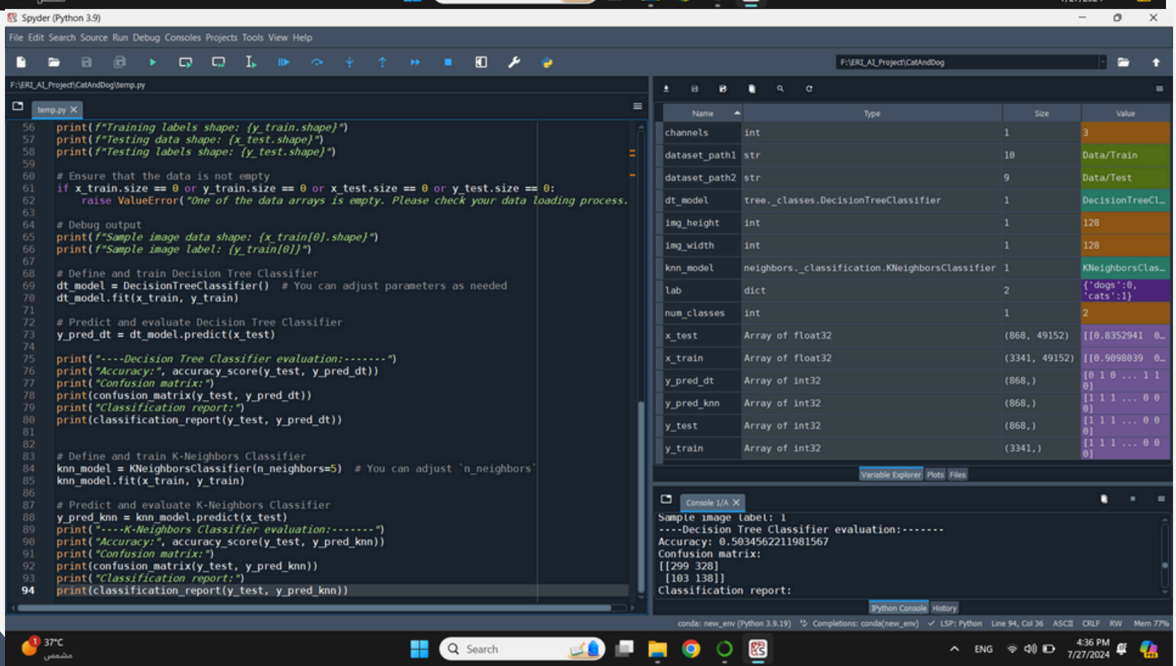
```python
dt_model = DecisionTreeClassifier()  # You can adjust parameters as needed
dt_model.fit(x_train, y_train)

# Predict and evaluate Decision Tree Classifier
y_pred_dt = dt_model.predict(x_test)

print("----Decision Tree Classifier evaluation:-------")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Confusion matrix:")
print(confusion_matrix(y_test, y_pred_dt))
print("Classification report:")
print(classification_report(y_test, y_pred_dt))


# Define and train K-Neighbors Classifier
knn_model = KNeighborsClassifier(n_neighbors=5)  # You can adjust `n_neighbors`
knn_model.fit(x_train, y_train)

# Predict and evaluate K-Neighbors Classifier
y_pred_knn = knn_model.predict(x_test)
print("----K-Neighbors Classifier evaluation:-------")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Confusion matrix:")
print(confusion_matrix(y_test, y_pred_knn))
print("Classification report:")
print(classification_report(y_test, y_pred_knn))
```

Console 1/A

```
Sample image label: 1
----Decision Tree Classifier evaluation:-------
Accuracy: 0.5034562211981567
Confusion matrix:
[[299 328]
 [103 138]]
Classification report:
              precision    recall  f1-score   support

           0       0.74      0.48      0.58       627
           1       0.30      0.57      0.39       241

    accuracy                           0.50       868
   macro avg       0.52      0.52      0.49       868
weighted avg       0.62      0.50      0.53       868

----K-Neighbors Classifier evaluation:-------
Accuracy: 0.4735023041474654
Confusion matrix:
[[222 405]
 [ 52 189]]
Classification report:
              precision    recall  f1-score   support

           0       0.81      0.35      0.49       627
           1       0.32      0.78      0.45       241

    accuracy                           0.47       868
   macro avg       0.56      0.57      0.47       868
weighted avg       0.67      0.47      0.48       868
```


Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
F:\ERI_AI_Project\CatAndDog\temp.py
temp.py
import os
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
# Set image parameters
img_height = 128
img_width = 128
channels = 3
num_classes = 2 # Adjusted to match the number of classes in lab dictionary
# Dictionary to label the images
lab = {'dogs': 0, 'cats': 1}
def load_images_from_folder(folder_path, label_dict):
data = []
labels = []
print(f"Loading images from folder: {folder_path}")
for folder in os.listdir(folder_path):
folder_path_full = os.path.join(folder_path, folder)
if not os.path.isdir(folder_path_full):
print(f"Skipping {folder_path_full} as it is not a directory.")
continue
print(f"Processing folder: {folder}")
for filename in os.listdir(folder_path_full):
file_path = os.path.join(folder_path_full, filename)
if os.path.isfile(file_path):
if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):
try:
img = load_img(file_path, target_size=(img_height, img_width))
img_array = img_to_array(img)
data.append(img_array.flatten()) # Flatten the image array
labels.append(label_dict[folder])
except Exception as e:
Name Type Size Value
channels int 1 3
Variable Explorer Plots Files
Console 1/A
Sample image label: 1
----Decision Tree Classifier evaluation:-------
Accuracy: 0.5034562211981567
Confusion matrix:
[[299 328]
[103 138]]
Classification report:
precision recall f1-score support
0 0.74 0.48 0.58 627
1 0.30 0.57 0.39 241
accuracy 0.50 868
macro avg 0.52 0.52 0.49 868
weighted avg 0.62 0.50 0.53 868
----K-Neighbors Classifier evaluation:-------
Accuracy: 0.4735023041474654
Confusion matrix:
[[222 405]
[ 52 189]]
Classification report:
precision recall f1-score support
0 0.81 0.35 0.49 627
1 0.32 0.78 0.45 241
accuracy 0.47 868
macro avg 0.56 0.57 0.47 868
weighted avg 0.67 0.47 0.48 868
In [4]:
IPython Console History
37°C
4:35 PM
7/27/2024


Spyder (Python 3.9)
except Exception as e:
print(f"Error loading image {file_path}: {e}")
else:
print(f"Skipping non-image file {file_path}.")
print(f"Loaded {len(data)} images from {folder_path}")
return np.array(data) / 255.0, np.array(labels)
# Load training data
dataset_path1 = 'Data/Train'
x_train, y_train = load_images_from_folder(dataset_path1, lab)
# Load testing data
dataset_path2 = 'Data/Test'
x_test, y_test = load_images_from_folder(dataset_path2, lab)
# Check if data is loaded correctly
print(f"Training data shape: {x_train.shape}")
print(f"Training labels shape: {y_train.shape}")
print(f"Testing data shape: {x_test.shape}")
print(f"Testing labels shape: {y_test.shape}")
# Ensure that the data is not empty
if x_train.size == 0 or y_train.size == 0 or x_test.size == 0 or y_test.size == 0:
raise ValueError("One of the data arrays is empty. Please check your data loading process.
# Debug output
print(f"Sample image data shape: {x_train[0].shape}")
print(f"Sample image label: {y_train[0]}")
# Define and train Decision Tree Classifier
dt_model = DecisionTreeClassifier() # You can adjust parameters as needed
dt_model.fit(x_train, y_train)
# Predict and evaluate Decision Tree Classifier
y_pred_dt = dt_model.predict(x_test)
print("----Decision Tree Classifier evaluation:-------")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Confusion matrix:")
4:35 PM
7/27/2024


Spyder (Python 3.9)
# Define and train K-Neighbors Classifier
knn_model = KNeighborsClassifier(n_neighbors=5) # You can adjust `n_neighbors`
knn_model.fit(x_train, y_train)
# Predict and evaluate K-Neighbors Classifier
y_pred_knn = knn_model.predict(x_test)
print("----K-Neighbors Classifier evaluation:-------")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Confusion matrix:")
print(confusion_matrix(y_test, y_pred_knn))
print("Classification report:")
print(classification_report(y_test, y_pred_knn))
Name Type Size Value
channels int 1 3
dataset_path1 str 10 Data/Train
dataset_path2 str 9 Data/Test
dt_model tree._classes.DecisionTreeClassifier 1
img_height int 1 128
img_width int 1 128
knn_model neighbors._classification.KNeighborsClassifier 1
lab dict 2 {'dogs':0, 'cats':1}
num_classes int 1 2
x_test Array of float32 (868, 49152)
x_train Array of float32 (3341, 49152)
y_pred_dt Array of int32 (868,)
y_pred_knn Array of int32 (868,)
y_test Array of int32 (868,)
y_train Array of int32 (3341,)
4:36 PM
7/27/2024

# Unsupervized ML

Used Histogram of Oriented Gradients (HOG) for feature extraction, Principal Component Analysis (PCA) for dimensionality reduction, and KMeans clustering for grouping similar images. The project evaluates the clustering results against the ground truth labels and visualizes the clusters and classification results. (How ever because the images are complex this approach wasn't the best for this case)

### Data Preprocessing

Image Loading and Preprocessing

Images are loaded from separate folders for training and testing datasets. Each image is resized to 64x64 pixels and normalized by dividing pixel values by 255.0. The labels are assigned as 0 for dogs and 1 for cats.

```
# Function to load images from a folder and preprocess them
def load_images_and_labels_from_folder(folder, image_size=(64, 64)):
    images = []
    labels = []
    for label, subfolder in enumerate(['dogs', 'cats']):
        subfolder_path = os.path.join(folder, subfolder)
        for filename in os.listdir(subfolder_path):
            if filename.endswith(".jpg") or filename.endswith(".png"):
                img_path = os.path.join(subfolder_path, filename)
                img = io.imread(img_path)
                if img is not None:
                    img = cv2.resize(img, image_size)
                    img = img / 255.0  # Normalize the image
                    images.append(img)
                    labels.append(label)  # 0 for dogs, 1 for cats
    return np.array(images), np.array(labels)

```

## HOG Feature Extraction

HOG features are extracted from the grayscale images. The HOG descriptor captures the edge and gradient structure of the images, which is useful for distinguishing between different object categories.

python

```python
# Extract HOG features from images
def extract_hog_features(images, pixels_per_cell=(16, 16), cells_per_block=(2, 2), visualize=False):
    hog_features = []
    for img in images:
        # Convert to grayscale
        img_gray = color.rgb2gray(img)
        if visualize:
            features, _ = hog(img_gray, pixels_per_cell=pixels_per_cell,
                             cells_per_block=cells_per_block, visualize=visualize)
        else:
            features = hog(img_gray, pixels_per_cell=pixels_per_cell,
                           cells_per_block=cells_per_block, visualize=visualize)
        hog_features.append(features)
    return np.array(hog_features)
```

## Standardization and Scaling

The extracted HOG features are standardized using StandardScaler and optionally scaled to a [0, 1] range using MinMaxScaler.

```python
# Standardize the HOG features
scaler_standard = StandardScaler()
train_hog_features_standard_scaled = scaler_standard.fit_transform(train_hog_features)
test_hog_features_standard_scaled = scaler_standard.transform(test_hog_features)

# Apply MinMaxScaler
scaler_minmax = MinMaxScaler(feature_range=(0, 1))
train_hog_features_minmax_scaled = scaler_minmax.fit_transform(train_hog_features)
test_hog_features_minmax_scaled = scaler_minmax.transform(test_hog_features)
```

## Dimensionality Reduction

PCA is applied to the standardized HOG features to reduce dimensionality and retain 50 principal components. This step helps in reducing the computational cost and noise while preserving the essential features for clustering.

```python
# Apply PCA for feature extraction
pca = PCA(n_components=50)
train_hog_features_pca = pca.fit_transform(train_hog_features_standard_scaled)
test_hog_features_pca = pca.transform(test_hog_features_standard_scaled)
print(f"Training PCA shape: {train_hog_features_pca.shape}")
print(f"Testing PCA shape: {test_hog_features_pca.shape}")
```

## Clustering

### Determining Optimal Number of Clusters

The elbow method is used to determine the optimal number of clusters by plotting the within-cluster sum of squares (inertia) for different values of k.

```python
# Determine the optimal number of clusters using the elbow method
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', n_init=10, random_state=0)
    kmeans.fit(train_hog_features_pca)
    inertia.append(kmeans.inertia_)

# Plot the elbow graph
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('Within-cluster Sum of Squares')
plt.show()
```

### KMeans Clustering

Based on the elbow graph, the optimal number of clusters is chosen, and KMeans clustering is applied to the PCA-transformed features.

```python
# Choose the optimal number of clusters based on the elbow graph
optimal_clusters = 2  # Set this to the number of clusters identified by the elbow method

# Apply KMeans clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', n_init=10, random_state=0)
train_clusters = kmeans.fit_predict(train_hog_features_pca)
test_clusters = kmeans.predict(test_hog_features_pca)
```

### Evaluation

The clustering results are evaluated against the ground truth labels using confusion matrices and classification reports. Clusters are relabeled to align with the actual labels for better interpretability.

```python
# Print evaluation metrics
print("Ground truth clusters:\n", test_labels)
print("Estimated clusters:\n", test_clusters)
print("Confusion matrix:\n", confusion_matrix(test_labels, test_clusters))
print("Classification report:\n", classification_report(test_labels, test_clusters))

# Relabel clusters
relabel_clusters = [1 if cluster == 0 else 0 for cluster in test_clusters]

print("Relabel clusters:\n", relabel_clusters)
print("Confusion matrix (relabelled):\n", confusion_matrix(test_labels, relabel_clusters))
print("Classification report (relabelled):\n", classification_report(test_labels,
    relabel_clusters))
```

## Visualization

### PCA Component Plot

The PCA components are plotted to visualize the clustering results

```
# Function to plot PCA components
def plot_pca_components(images_pca, labels, title):
    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(images_pca[:, 0], images_pca[:, 1], c=labels, cmap='viridis', alpha=0.5)
    plt.title(title)
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.colorbar(scatter, label='Cluster')
    plt.show()

# Plot PCA components of training data
plot_pca_components(train_hog_features_pca, train_clusters, 'KMeans Clustering Results (Training Data)')

```

### Image Plot with Cluster Labels

A subset of test images is plotted with their predicted cluster labels.

```
# Function to plot images with cluster labels
def plot_images(images, labels, title, image_size=(64, 64)):
    fig, axes = plt.subplots(2, 5, figsize=(12, 6))
    fig.suptitle(title)
    for i, ax in enumerate(axes.flat):
        if i < len(images):
            ax.imshow(images[i].reshape(image_size + (3,)))
            ax.set_title(f'Cluster {labels[i]}')
        ax.axis('off')
    plt.show()

# Plot images with predicted clusters
plot_images(test_images, test_clusters, 'KMeans Clustering Results (Test Data)')

```

### 2D PCA Visualization

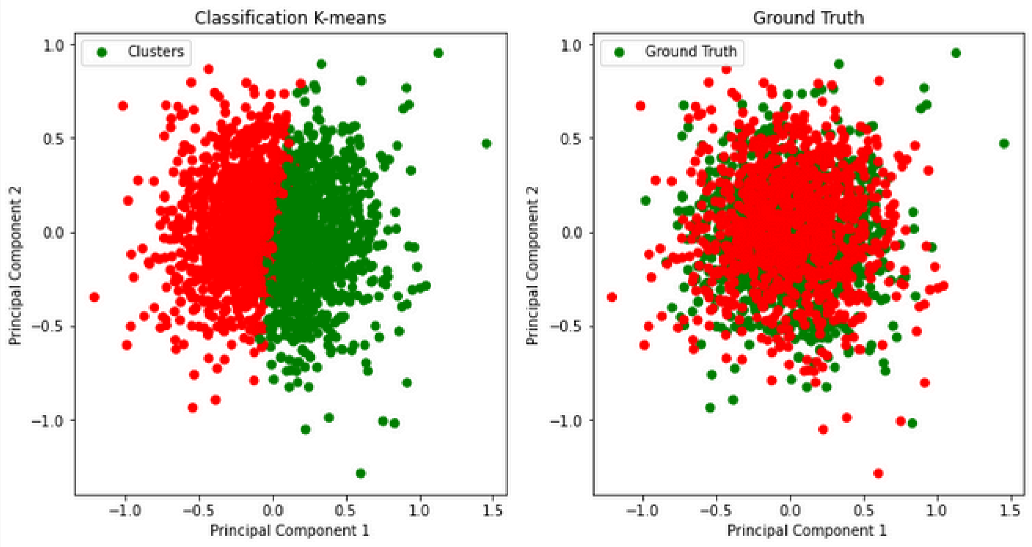
PCA is performed again to reduce the dimensionality to 2D for visualization of clustering results and ground truth labels.

```
# Perform PCA to reduce dimensionality to 2D for visualization
pca_2d = PCA(n_components=2)
pca_result = pca_2d.fit_transform(test_hog_features)  # Use HOG features for PCA visualization

# Visualize the clustering results
colormap = np.array(['green', 'red'])

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.title('Classification K-means')
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=colormap[relabel_clusters], label='Clusters')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()

plt.subplot(1, 2, 2)
plt.title('Ground Truth')
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=colormap[test_labels], label='Ground Truth')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()

plt.show()

```

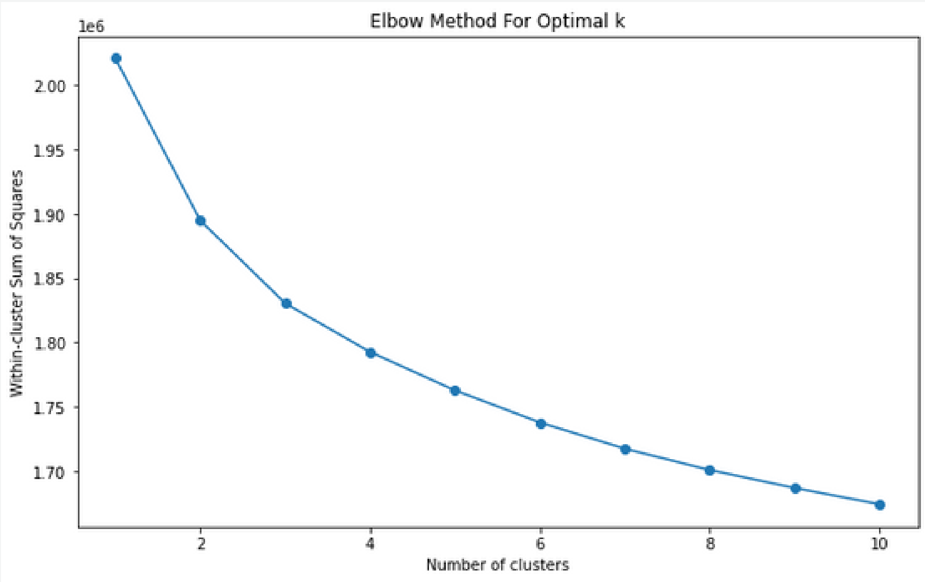
## Output:

```
In [1]: runfile('C:/Users/Salma/SpyderProjects/AIcourse/Unsupervised ML/scripts/main.py', wdir='C:/Users/Salma/SpyderProjects/
AIcourse/Unsupervised ML/scripts')
Training images shape: (8005, 64, 64, 3)
Training labels shape: (8005,)
Testing images shape: (2023, 64, 64, 3)
Testing labels shape: (2023,)
Training PCA shape: (8005, 50)
Testing PCA shape: (2023, 50)

                                   Important
 Figures are displayed in the Plots pane by default. To make them also appear inline in the console, you need to
 uncheck "Mute inline plotting" under the options menu of Plots.

 Ground truth clusters:
 [0 0 0 ... 1 1 1]
Estimated clusters:
 [1 1 0 ... 1 1 1]
Confusion matrix:
 [[552 460]
 [537 474]]
Classification report:
               precision    recall  f1-score   support

           0       0.51      0.55      0.53      1012
           1       0.51      0.47      0.49      1011

    accuracy                           0.51      2023
   macro avg       0.51      0.51      0.51      2023
weighted avg       0.51      0.51      0.51      2023

 Relabel clusters:
  [0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0,
 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1,
 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0,
 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0,
 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0,
 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0,
 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1,
 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0,
 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1,
 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0,
 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0,
 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0,
 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0,
 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0,
 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0]
 Confusion matrix (relabelled):
  [[460 552]
  [474 537]]
 Classification report (relabelled):
               precision    recall  f1-score   support

           0       0.49      0.45      0.47      1012
           1       0.49      0.53      0.51      1011

    accuracy                           0.49      2023
   macro avg       0.49      0.49      0.49      2023
weighted avg       0.49      0.49      0.49      2023
```

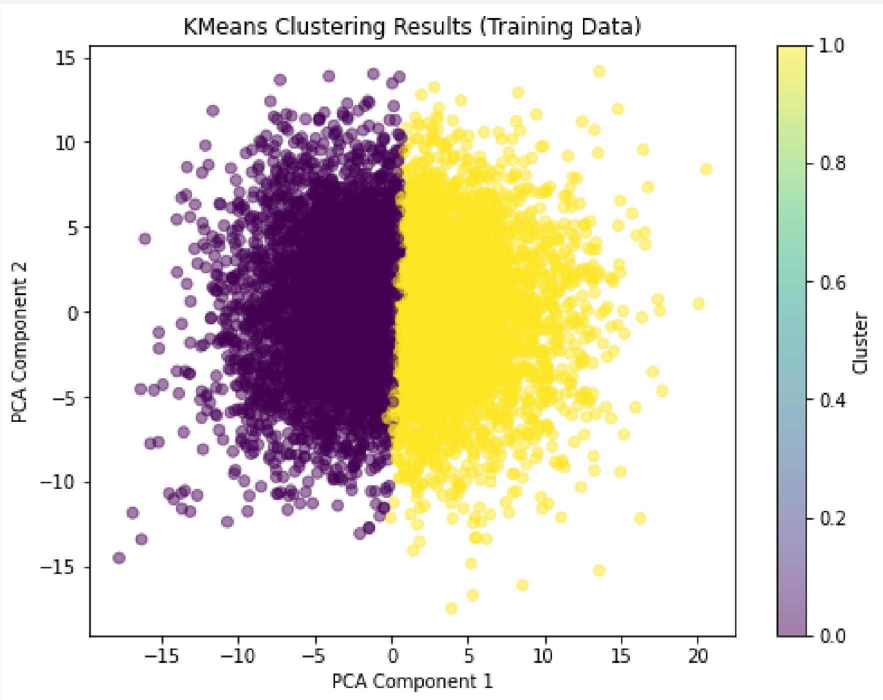
## Classfication KMeans and Ground Truth



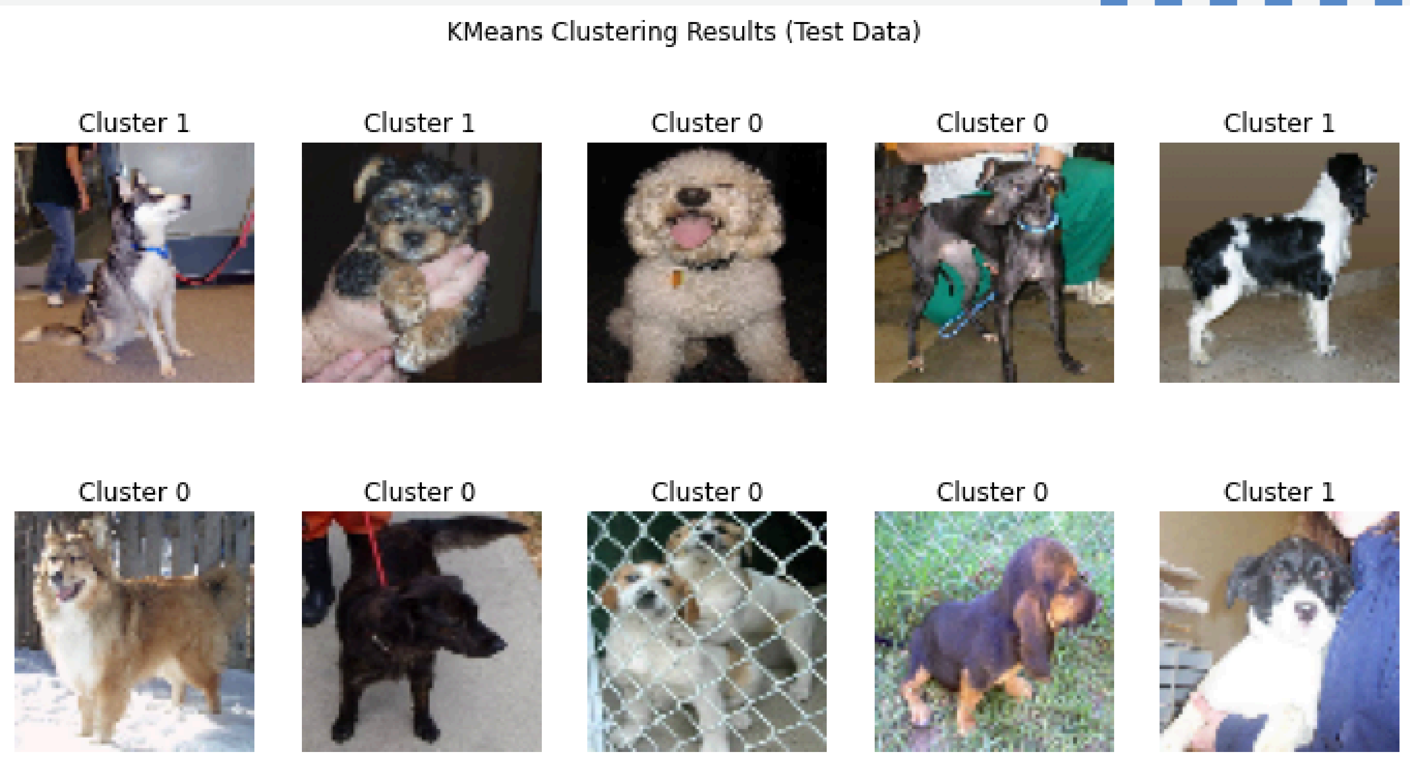

## Elbow method for optimal K




## KMeans Clustering Results (Trainging Data)



## KMeans Clustering Results (Test Data)




# Conclusion

The application of unsupervised learning techniques for image classification. The combination of HOG features, PCA, and KMeans clustering provides a method for grouping similar images without the need for labeled training data. The results, evaluated through confusion matrices and visualizations, show the effectiveness of this approach in distinguishing between images of dogs and cats. Future improvements could include experimenting with different feature extraction methods, clustering algorithms, and further hyperparameter tuning to enhance performance.

# Deep Learning

Deep learning is a subset of machine learning that uses multilayered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain..

### Data Preprocessing

Image Loading and Preprocessing

Images are loaded from separate folders for training and testing datasets. Each image is resized to 64x64 pixels and normalized by dividing pixel values by 255.0. The labels are assigned as 0 for dogs and 1 for cats.

```
# Load training data
train_data = []
train_labels = []
train_base_path = 'C:/Users/t460/Downloads/archive/training_set/training_set'

for folder in os.listdir(train_base_path):
    folder_path = os.path.join(train_base_path, folder)
    if os.path.isdir(folder_path):
        for filename in os.listdir(folder_path):
            img_path = os.path.join(folder_path, filename)
            try:
                img = load_img(img_path, target_size=(img_height, img_width))
                img_array = img_to_array(img)
                train_data.append(img_array)
                train_labels.append(lab[folder])
            except Exception as e:
                print(f"Error loading image {img_path}: {e}")
```

This section of the code focuses on preparing and normalizing the training data for a machine learning model. It converts the training data into a NumPy array and normalizes the pixel values by dividing by 255.0, which scales the values to a range between 0 and 1. This is a common preprocessing step for image data to ensure that the model trains more effectively. Additionally, the labels for the training data are also converted into a NumPy array. The code then proceeds to load the test data and labels from the specified directory path, preparing them for evaluation purposes.

```
x_train = np.array(train_data) / 255.0  # Normalize pixel values
y_train = np.array(train_labels)

# Load test data
test_data = []
test_labels = []
test_base_path = 'C:/Users/t460/Downloads/archive/test_set/test_set'
```

This section of the code is responsible for loading and preprocessing the test data for the machine learning model. It iterates through each folder in the specified test data directory. For each folder, it checks if the path is a directory and then iterates through each image file within that directory. Each image is loaded and resized to the specified dimensions (`img_height` and `img_width`). The image is then converted into an array and appended to the `test_data` list. Correspondingly, the label for the image, determined by the folder name, is appended to the `test_labels` list. If there is an error loading an image, it prints an error message. Finally, the test data and labels are converted into NumPy arrays, and the pixel values of the test data are normalized by dividing by 255.0, ensuring they are in the range of 0 to 1.

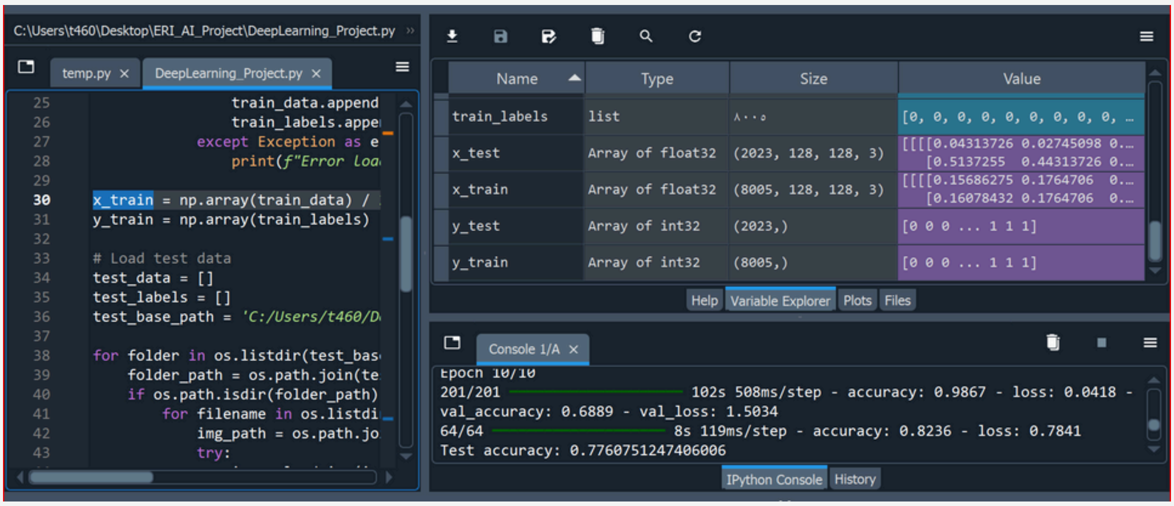
```
for folder in os.listdir(test_base_path):
    folder_path = os.path.join(test_base_path, folder)
    if os.path.isdir(folder_path):
        for filename in os.listdir(folder_path):
            img_path = os.path.join(folder_path, filename)
            try:
                img = load_img(img_path, target_size=(img_height, img_width))
                img_array = img_to_array(img)
                test_data.append(img_array)
                test_labels.append(lab[folder])
            except Exception as e:
                print(f"Error loading image {img_path}: {e}")

x_test = np.array(test_data) / 255.0  # Normalize pixel values
y_test = np.array(test_labels)
```

This section of the code defines, trains, and evaluates a Convolutional Neural Network (CNN) model for image classification:

1. Model Definition:
   - The model is built using convolutional (Conv2D) and pooling (MaxPooling2D) layers to extract features from images.
   - A flattening layer (Flatten) is used to convert the 2D matrix data to a vector, followed by dense (Dense) layers for classification.
2. Model Compilation:
   - The model is compiled with the sparse_categorical_crossentropy loss function and adam optimizer.
3. Model Training:
   - The model is trained on the training data for 10 epochs with a batch size of 32, using 20% of the data for validation.
4. Model Evaluation:
   - The model's accuracy is evaluated on the test data, and the test accuracy is printed.

```python
# Define the model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accurac
# Print model summary
model.summary()
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc}")
```

The accuracy



Our

# Team

**shahd waleed** **Bassant Tamer**

**Salma Osama** **Fayza Mahmoud**