Addis Ababa University

Master's in Artificial Intelligence

Computer Vision (CV) Laboratory Manual

Prepared by: Dr. Natnael Argaw Wondimu

Environment: OpenCV and Visual Studio

Preface

This manual is prepared in a workbook style for AI Master's students at Addis Ababa University. It provides a hands-on introduction to Computer vision with OpenCV, covering advanced topics. Each lab includes objectives, theoretical background, procedures, OpenCV code, checkpoints, try-it-yourself prompts, and collaborative assignments. The manual is intended to serve as a base for advanced studies in Computer Vision.

Chapter 8: Intro to Machine Learning (OpenCV + Scikit-learn – Python)

Objective

To introduce core supervised machine learning algorithms—K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Decision Trees—using Scikit-learn. Students will train, evaluate, and visualize simple classification models and integrate OpenCV for data processing.

1. What is Machine Learning?

Description: Machine learning (ML) is the science of training models to learn patterns from data. In computer vision, ML enables automated image classification, object detection, and more.

2. Libraries Setup

pip install scikit-learn opencv-python matplotlib numpy

Import Required Packages

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

3. Dataset Preparation (Digits Dataset)

```
from sklearn.datasets import load_digits

data = load_digits()
X = data.images
y = data.target

# Flatten the 8x8 images into 64 feature vectors
n_samples = len(X)
X = X.reshape((n_samples, -1))
```

Output Description: Loads a digit recognition dataset with 8x8 pixel images and flattens each image into a feature vector.

4. Splitting the Dataset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

Output Description: Splits the dataset into 75% training and 25% testing sets.

5. K-Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
```

```
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

Evaluation

```
print("KNN Classification Report:\n", classification_report(y_test, y_pred_knn))
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
```

Output Description: Provides classification accuracy and metrics for the KNN model.

6. Support Vector Machine (SVM)

```
from sklearn.svm import SVC

svm = SVC(kernel='linear', C=1)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
```

Evaluation

```
print("SVM Classification Report:\n", classification_report(y_test, y_pred_svm))
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
```

Output Description: Evaluates SVM classifier performance on the digit test set.

7. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier(max_depth=10)
dtree.fit(X_train, y_train)
y_pred_tree = dtree.predict(X_test)
```

Evaluation

```
print("Decision Tree Report:\n", classification_report(y_test, y_pred_tree))
print("Accuracy:", accuracy_score(y_test, y_pred_tree))
```

Output Description: Evaluates decision tree model accuracy and classification ability.

8. Confusion Matrix Visualization

```
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - KNN')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Output Description: Displays confusion matrix heatmap for KNN classifier results.

9. Integration with OpenCV (Optional)

Use OpenCV for real-world input data and feed into scikit-learn classifiers.

```
img = cv2.imread('digit_sample.png', 0)
img = cv2.resize(img, (8, 8))
img = 16 - (img // 16)  # Normalize to 0–16 range
img_flat = img.flatten().reshape(1, -1)
predicted_digit = knn.predict(img_flat)
print("Predicted Digit:", predicted_digit[0])
```

Output Description: Processes and classifies a new digit image using a trained model.

10. Summary

- **KNN**: Simple and effective for small datasets.
- **SVM**: Excellent generalization with margin maximization.
- **Decision Trees**: Easy to interpret; prone to overfitting without pruning.
- Use **scikit-learn** for model training and **OpenCV** for image preprocessing.

Suggested Exercises

1. Compare models using cross-validation.
2. Tune hyperparameters (e.g., K in KNN, max_depth in trees).
3. Train models using your own digit/character datasets via OpenCV.
4. Build a GUI-based digit recognition app using OpenCV and a trained model.