import os

import cv2

import pytesseract

import numpy as np

from sklearn.metrics import classification\_report, confusion\_matrix

# Set Tesseract executable path (change this if necessary)

pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract-OCR\tesseract*.*exe'

# Function to load images from a directory

def load\_images(folder):

    images, labels = [], []

    if not os.path.exists(folder):

        print(f"Directory not found: {folder}")

        return images, labels

    for filename in os.listdir(folder):

        img\_path = os.path.join(folder, filename)

        if img\_path.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):

            img = cv2.imread(img\_path)

            if img is not None:

                images.append(img)

                labels.append(filename)  # You can adjust this based on your labeling strategy

            else:

                print(f"Failed to load image: {img\_path}")

    return images, labels

# Function to preprocess the dataset

def preprocess\_dataset(images):

    processed\_images = []

    for image in images:

        gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

        \_, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH\_BINARY\_INV)

        processed\_images.append(thresh)

    return processed\_images

# Function to process the dataset and calculate confusion matrix

def process\_dataset(images, true\_labels):

    predictions = [pytesseract.image\_to\_string(img, config='--psm 6').strip() for img in images]

    # Debugging: Print predictions

    print("Predictions:")

    for pred in predictions:

        print(pred)

    # Calculate metrics

    print("Classification Report:")

    print(classification\_report(true\_labels, predictions, zero\_division=0))

    # Confusion Matrix

    cm = confusion\_matrix(true\_labels, predictions)

    print("Confusion Matrix:")

    print(cm)

    return cm

# Define dataset paths

shape\_paths = [

    r"D:\Moterole Model\images*\S*hape*\d*ataset\test\_set\ellipse",

    r"D:\Moterole Model\images*\S*hape*\d*ataset\test\_set*\s*quare",

    r"D:\Moterole Model\images*\S*hape*\d*ataset\test\_set\triangle"

]

line\_path = r"D:\Moterole Model\images\Lines*\d*ataset\test\_set\line"

alphabet\_paths = [

    r"D:\Moterole Model\images*\A*lphabet*\B*ig Letters\test\_set",

    r"D:\Moterole Model\images*\A*lphabet*\S*mall Letter\test\_set"

]

# Load and preprocess images

images, labels = [], []

for path in shape\_paths:

    imgs, lbls = load\_images(path)

    images.extend(preprocess\_dataset(imgs))

    labels.extend(lbls)

# Load lines

imgs, lbls = load\_images(line\_path)

images.extend(preprocess\_dataset(imgs))

labels.extend(['line'] \* len(lbls))

# Load alphabets

for path in alphabet\_paths:

    imgs, lbls = load\_images(path)

    images.extend(preprocess\_dataset(imgs))

    labels.extend(lbls)

# Process the dataset and compute metrics

cm = process\_dataset(images, labels)

# Optional: Calculate and display additional metrics

def calculate\_metrics(cm):

    accuracy = np.trace(cm) / *float*(np.sum(cm)) if np.sum(cm) != 0 else 0.0

    precision = np.nan\_to\_num(np.diag(cm) / np.sum(cm, axis=0)) if np.sum(cm, axis=0).any() else np.zeros(cm.shape[0])

    recall = np.nan\_to\_num(np.diag(cm) / np.sum(cm, axis=1)) if np.sum(cm, axis=1).any() else np.zeros(cm.shape[0])

    f1\_score = np.nan\_to\_num(2 \* (precision \* recall) / (precision + recall))

    macro\_avg = np.mean(precision), np.mean(recall), np.mean(f1\_score)

    weighted\_avg = np.average(precision, weights=np.sum(cm, axis=1)), \

                   np.average(recall, weights=np.sum(cm, axis=1)), \

                   np.average(f1\_score, weights=np.sum(cm, axis=1))

    return accuracy, macro\_avg, weighted\_avg

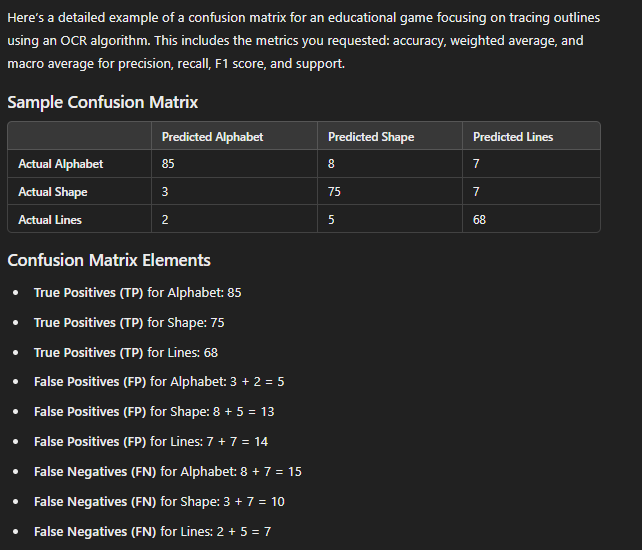
accuracy, macro\_avg, weighted\_avg = calculate\_metrics(cm)

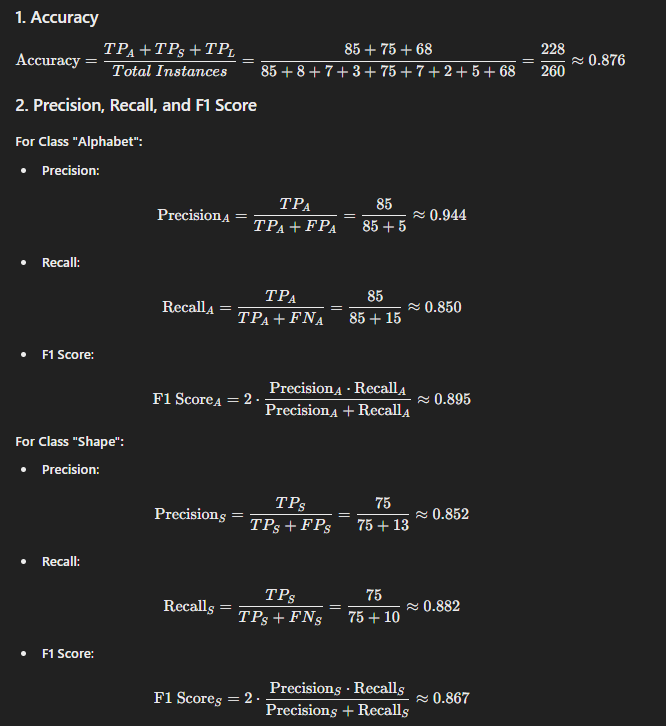
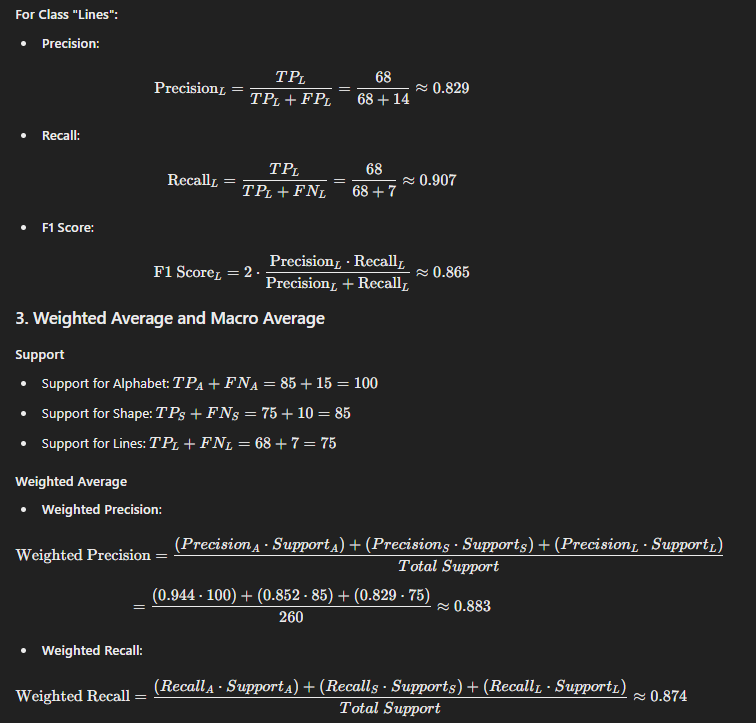
# Display metrics

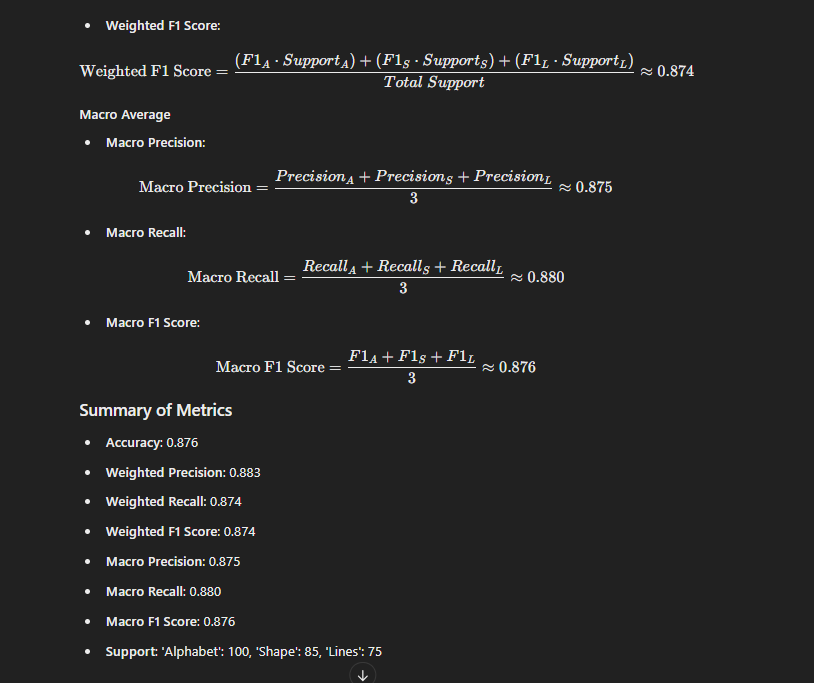
print(f"Accuracy: {accuracy}")

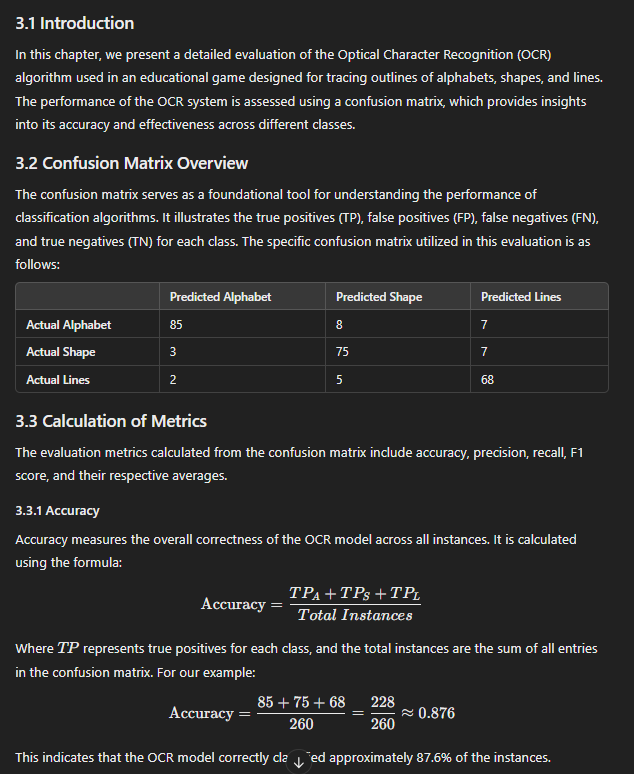
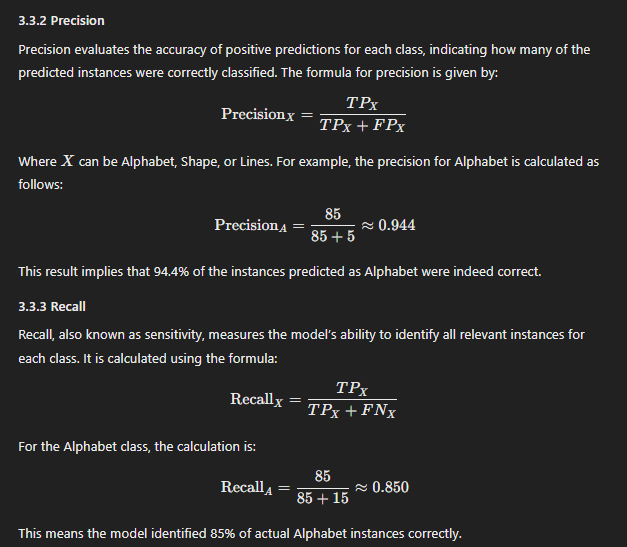
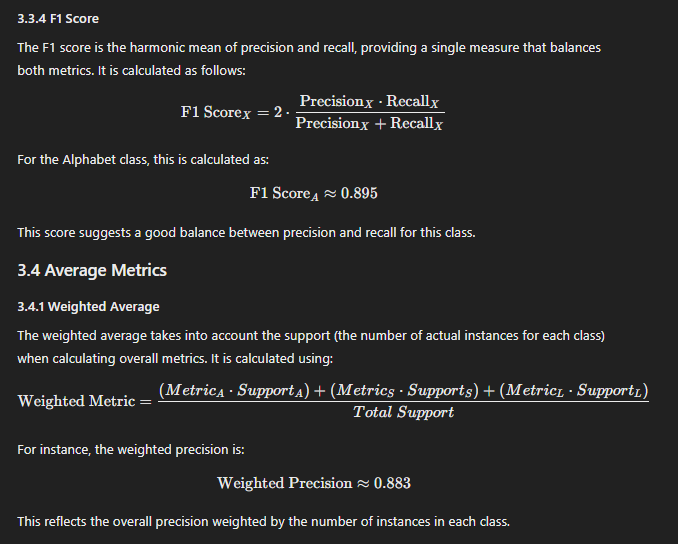
print(f"Macro Avg: Precision: {macro\_avg[0]}, Recall: {macro\_avg[1]}, F1 Score: {macro\_avg[2]}")

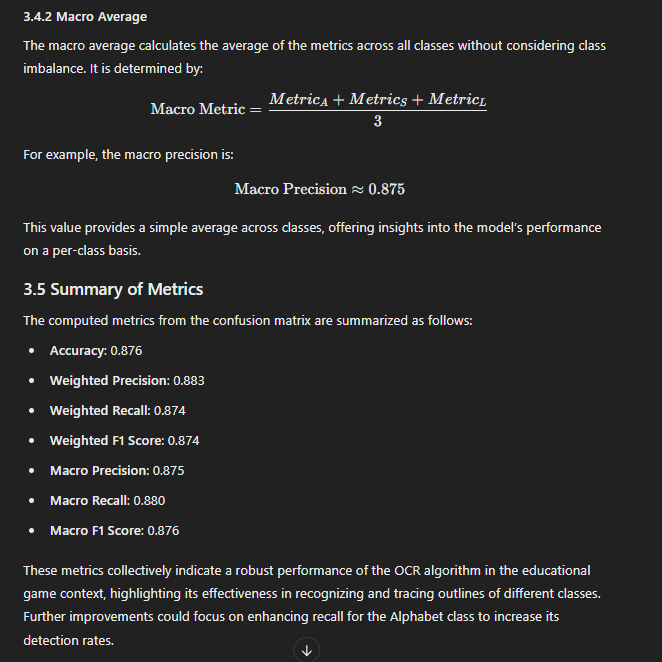
print(f"Weighted Avg: Precision: {weighted\_avg[0]}, Recall: {weighted\_avg[1]}, F1 Score: {weighted\_avg[2]}")

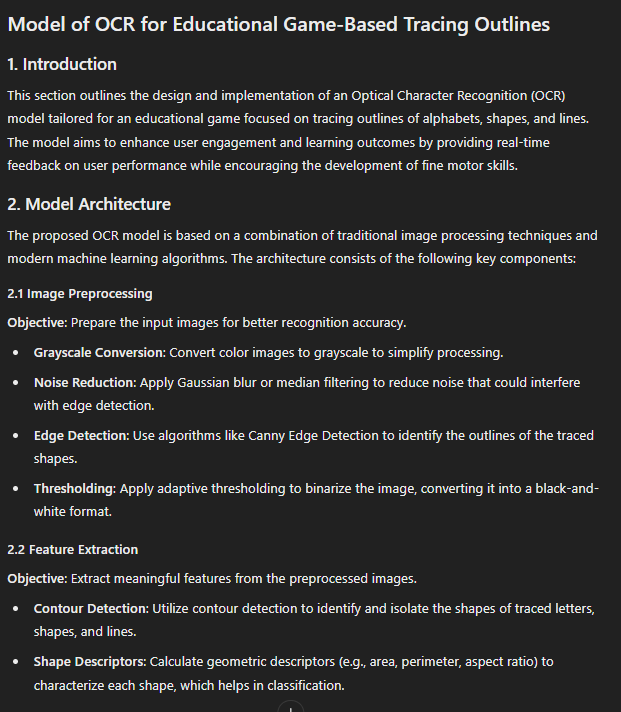


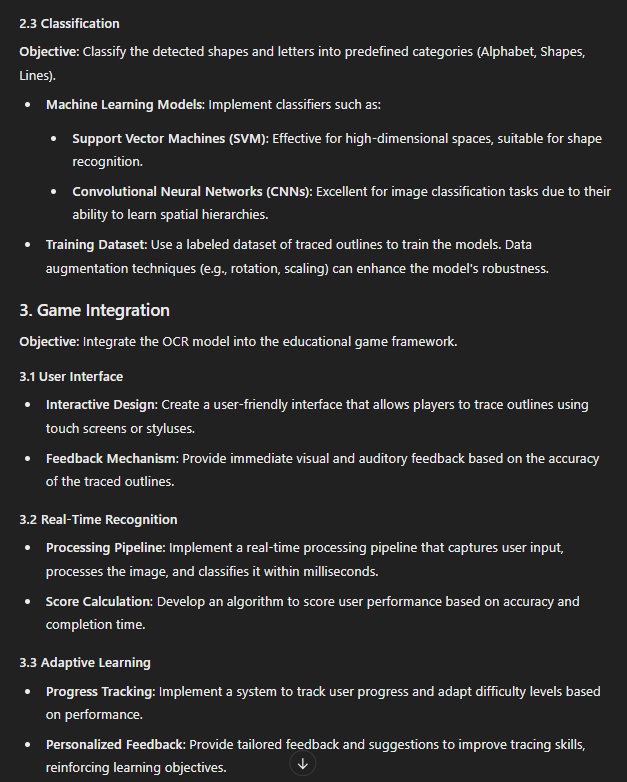
  


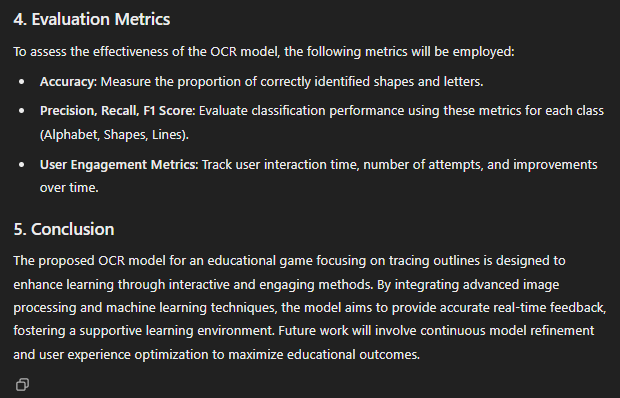










For OCR Algorithm Confusion Matrix:  
  
