Proyecto Final de Visión por Computadora

Descripción

Esta aplicación permite aplicar filtros de procesamiento de imágenes en tiempo real usando OpenCV y una interfaz gráfica construida con PySimpleGUI.

Estructura del Proyecto

```
vision_computer_project/
├── modulos_proyecto_3P         # Carpeta para almacenar los módulos utilizados
     ├── circles_detect.py           #Funciones para la detección de circulos
     ├── color_range.py           # Funciones para definir rangos de colores
     ├── interface.py          # Código para la interfaz gráfica (GUI)
     ├── model_detect.py    # Código para ajustar los parametros de detección
     ├── model_trained.py     # Código para cargar el modelo entrenado
     ├── segmentation.py          # Funciones para la segmentación de imagen
├── cnn_model.pth             # Modelo CNN previamente entrando en Google Colab
├── principal.py               # Código principal para correr la app
├── requirements.txt      # Dependencias del proyecto
├── .gitignore            # Archivos y carpetas a ignorar por Git
└── README.md             # Instrucciones y objetivos del proyecto
```

Instalación

1. Ingresar a la carpeta correspondiente:

```
cd ./vision_computer_project/  # En Windows: .\vision_computer_project\
```

2. Clonar el repositorio.
3. Crear y activar un entorno virtual:

```
python -m venv venv
source venv/bin/activate  # En Windows: venv\Scripts\activate
```

4. Instalar dependencias:

```
pip install -r requirements.txt
```

Uso del Programa

Ejecutar la aplicación con:

```
python principal.py
```

Seleccionar el tipo de segmentación que se desea aplicar, la detección de círculos o si se desea utilizar el modelo CNN para detectar imágenes, y observa los resultados en tiempo real.

Extensiones

- Agrega diferentes segmentaciones en `segmentation.py`.
- Implementa detección de círuclos en `cricledetect.py`.
- Implementa rangos de colores en `colorange.py`.
- Personaliza la interfaz en `interface.py`.

Informe de Proyecto de Proyecto Final

Explicación del Funcionamiento del Código de los Diferentes Módulos del Proyecto

- **`Funcionamiento del código en el módulo circle_detect.py:`** Ver video de explicación aquí
- **`Funcionamiento del código en el módulo color_range.py:`** Ver video de explicación aquí
- **`Funcionamiento del código en el módulo interface.py:`** Ver video de explicación aquí
- **`Funcionamiento del código en el módulo model_trained.py:`** Ver video de explicación aquí
- **`Funcionamiento del código en el módulo model_detect.py:`** Ver video de explicación aquí
- **`Funcionamiento del código en el módulo segmentations.py:`** Ver video de explicación aquí
- **`Funcionamiento del código en el módulo principal.py:`** Ver video de explicación aquí

Explicación de la Ejecución del Programa y el Funcionamiento de su Interfaz

- **`Funcionamiento de la interfaz y la proceso de ejecución correcta del programa:`** Ver video de explicación aquí

Explicación de cada Módulo

1. Módulo `circles_detect.py`

Este módulo implementa la clase `CircleDetector`, que se va a encargar de ayudar en la detección círculos en las imágenes segmentadas utilizando el algoritmo de Hough.

Funcionalidades Clave:

- `detect_circles`: Se aplica mediante el algoritmo de Hough para detectar los círculos de acuerdo a la configuración de la segmentación, esto se realiza mediante el uso de las técnicas de morfología para

mejorar la calidad de la imagen.

- `draw_detected_circles`: Se encarga de dibujar los círculos detectados despues de realizar la segmentación, lo que permite al usuario visualizar correctamente los circulos segmentados una vez ya dibujados.

Decisiones de Diseño: Se eligió el algoritmo de Hough debido a que resultó ser el más sencillo de aprender a utilizar para poder aplicarlo en la detección de formas circulares.

2. Módulo `color_range.py`

Este módulo implementa la clase `ColorRange`, que se encarga de definir y gestionar los rangos de color definidos para la segmentación.

Funcionalidades Clave:

- `set_color_ranges`: Se encarga de establecer los rangos de color bajo y alto en el espacio de color utilizado, en este caso fue para HSV.
- `get_color_ranges`: Este se encarga de devolver los rangos de colores que se han predefinido para los diferentes colores que se pudieron segmentar.

Decisiones de Diseño: Se optó por definir estas funciones de esta manera, ya que eran las que permitían definir eficientemente el rango de colores y aplicar los rangos de colores que se predefinió.

3. Módulo `interface.py`

Este módulo se encarga de definir la interfaz gráfica de usuario (GUI) y fue hecha utilizando PySimpleGUI, que debido a las configuraciones que se asígnaron, permite seleccionar entre diferentes segmentaciones, ajustar los valores de las segmentaciones, aplicar el detector de círuclos y el modelo CNN.

Funcionalidades Clave:

- **Interfaz Intuitiva**: Proporciona una interfaz gráfica e interactiva fácil de usar para la selección de opciones de segmentación y detección.
- **Controles Interactivos**: Aquí se incluyen controles deslizantes para ajustar parámetros en tiempo real y casillas de verificación para seleccionar la funcion que se desea utilizar.

Decisiones de Diseño: Se utilizó PySimpleGUI por su simplicidad y facilidad de uso, lo que permitió crear la interfaz gráfica del programa de una manera rápida, sencilla y efectiva.

4. Módulo `model_detect.py`

Este módulo implementa la clase `ModelDetector`, que se encarga de cargar el modelo de red neuronal y realizar predicciones sobre las imágenes que se coloquen en frente de la cámara.

Funcionalidades Clave:

- `predict`: Aquí se toma el frame de la cámara de video y se encarga de aplicar las transformaciones necesarias, también, utiliza el modelo CNN para poder predecir la clase del objeto en la imagen.

- **Transformaciones de Imagen**: Aquí se aplican las transformaciones necesarias para preparar la imagen para realizar la predicción de la clase del objeto con exito.

Decisiones de Diseño: Se optó por este modelo CNN que previamente fue entrenado en google colab, para poder realizar la detección de objetos y este modelo proporciona una precisión decente en la clasificación y detección de objetos.

5. Módulo `model_trained.py`

Este módulo define la arquitectura de la red neuronal convolucional (CNN) que es utilizada en el modelo para la clasificación de objetos.

Funcionalidades Clave:

- **Definición de la Red**: Aquí se establece la estructura de la red neuronal, lo que incluye las capas convolucionales y completamente conectadas.
- **Carga del Modelo**: Aquí es donde se carga el modelo CNN que va a ser utilizado en la clasificación de objetos.

Decisiones de Diseño: Se pudo notar que la arquitectura se diseñó para ser simple pero efectiva, y que utiliza capas convolucionales para extraer características de las imágenes.

6. Módulo `segmentation.py`

Este módulo implementa la clase `SegmentationOpenCV`, que se va a encargar de definir los diferentes algoritmos segmentación de imágenes utilizados.

Funcionalidades Clave:

- `apply_color_segmentation`: Realiza la segmentación de color en función del espacio de color seleccionado (HSV), aquí sus valores son modificables y se realiza mediante los sliders definidos en el módulo intarce.py.
- `apply_border_detector`: Se encarga de aplicar un detector de bordes a la imagen segmentada.
- `draw_detected_circles`: Aquí se dibujan los círculos que van a ser detectados detectados en la imagen.
- `apply_kmeans_segmentation`: Se va a encargar de aplicar la segmentación avanzada de kmeans.

Decisiones de Diseño: Se decidió utilizar las diferentes técnicas de segmentación aprendidas en clase y se realizó basandose en el espacio de color HSV, también, se hizo uso de un detector de bordes para poder visualizar de mejor manera los bordes que tiene la imagen.

7. Módulo `principal.py`

El módulo `principal.py` este sería el punto de entrada del programa, aquí se integran todos los módulos que componen programa y que permite controlar el funcionamiento de este mismo. Su función principal es capturar el video de la cámara web en tiempo real, aplicar los diferentes algoritmos de segmentación, el detector de círculos, aplicar el modelo CNN y mostrar los resultados mediante la interfaz gráfica de usuario.

Funcionalidades Clave:

- `Captura de Video`: Aquí se utilizó cv2.VideoCapture(0) para lograr acceder a la cámara y capturar video en tiempo real, lo que permitirá hacer uso de las funcionalidades del programa.
- `Interacción con el Usuario`: Ocurre mediante la interfaz gráfica, esta va a permitir al usuario la posibilidad de seleccionar entre las diferentes funcionalidades que estan disponibles.
- `Integración de Módulos`: Aquí se coordina la interacción entre los módulos cómo: segmentation.py, circles_detect, color_range.py, y model_detect.py, lo que facilitará la actualización del programa.
- `Predicción en Tiempo Real`: Cuando se activa el modelo CNN, este se va a encargar de realizar las predicciones sobre los frames de la cámara web utilizada, lo que permite al usuario detectar diferentes y clasificar objetos utiliando su cámara web.

Decisiones de Diseño: Se optó por eligir las funciones disponibles en OpenCV por su robustez en procesamiento de imágenes que ofrecen y PySimpleGUI por su facilidad de uso. La modularidad del diseño permite que cada componente funcione de manera independiente, facilitando la identificación de errores y la implementación de nuevas funcionalidades.

Recomendaciones

Para mejorar el rendimiento del sistema, se podría implementar técnicas de aprendizaje más avanzadas de segmentación y detección, y mejorar el modelo utilizado. Por último, se recomienda realizar pruebas que ocurran en diferentes condiciones de iluminación y con diversos tipos de objetos para evaluar la robustez del programa así como lo he ido realizando en el proceso de desarrollo de este mismo.

Conclusión

En conclusión, este proyecto ha sido diseñado mediante un enfoque modular, lo que permite que el proyecto sea facilmente expandible con nuevos módulos, la posibilidad de brindar un eficiente mantenimiento y una mejor optimización de estos módulos. Para la creación de los diferentes módulos fue necesario aplicar todos los conocimientos en python y los que fueron recibidos en las clases del segundo y tercer parcial, estos fueron: la introducción a machine learning, los algoritmos de segmentacion básicos y avanzados utilizadas en machine learning y el procedimiento de cómo se debe entrenar un modelo de detección de imágenes. Una vez integrados estos módulos, ayudan a proporcionar una solución efectiva para actividades en la que sea necesaria la detección y segmentación de objetos en tiempo real, lo que ayuda a ofrecer una experiencia fluida y eficiente al usuario que lo utilice. Por último, con ayuda de las recomendaciones que se han propuesto, espero que el programa pueda ser mejorado y adaptado a las futuras necesidades de los futuros usuarios.

Informe del Funcionamiento de la Desfragmentación y de la Función ReLU() en Python

Introducción

Si nos referimos al procesamiento de datos y al aprendizaje automático, la desfragmentación en python y la función ReLU (Rectified Linear Unit), estos son componentes muy fundamentales y de gran utilidad, en el caso de la desfragmentación, esta se refiere a la optimización de estructuras de datos para mejorar el rendimiento

en Python, mientras que la función ReLU es una función de activación muy utilizada en redes neuronales artificiales. Por esta razón, en este informe se va a explicar cúal es la definición, cómo es el funcionamiento de la desfragmentación en python y la función ReLU, además, de agregar un ejemplo en la que se logre explicar la aplicación de cada una de ellas.

La Desfragmentación en Python

Definición de la Desfragmentación

La desfragmentación de variables en Python consiste en optimizar la utilización de estructuras de datos para mejorar el rendimiento. Esto incluye la eliminación de datos innecesarios, la reestructuración de listas o arrays y la optimización de la forma en que se almacenan y acceden a los datos.

Funcionamiento de la Desfragmentación

El proceso de desfragmentación en Python se realiza mediante los siguientes pasos, que van a ser de gran utilidad si se desea mejorar el rendimiento:

- **Liberación de memoria:** Puede servir para eliminar variables y datos, estos resultan no ser tan necesarios y de esta manera se puede llegar a liberar espacio en la memoria.
- **Reestructuración de datos:** Ayuda a reorganizar listas, arrays u otras estructuras de datos para mejorar la eficiencia del acceso y almacenamiento.
- **Compactación de memoria:** Permite ajustar el uso de la memoria para que los datos estén almacenados de manera continua y de esta manera se estaría ayudando a reducir la fragmentación.

Ejemplo de Desfragmentación: Segmentación de Imágenes con K-means

```
import cv2
import numpy as np

class ImageSegmenter:
    def __init__(self, frame):
        self.frame = frame
        self.k_kmeans = None
        self.segmented_image = None

    def apply_kmeans_segmentation(self, k):
        self.k_kmeans = k
        if self.frame is None:
            print("Error: No se ha recibido un frame válido.")
            return None

        # Reformatear la imagen a un array 2D
        Z = self.frame.reshape((-1, 3))  # Cambiar a 3 para los canales de color
        Z = np.float32(Z)

        # Definir criterios para K-means
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)

        # Aplicar K-means
        _, labels, centers = cv2.kmeans(Z, self.k_kmeans, None, criteria, 10,
```

```
    cv2.KMEANS_RANDOM_CENTERS)

            # Convertir centros a uint8
            centers = np.uint8(centers)

            # Crear la imagen segmentada
            self.segmented_image = centers[labels.flatten()].reshape(self.frame.shape)

            return self.segmented_image

    # Ejemplo de uso
    # Cargar una imagen
    frame = cv2.imread('ruta_de_tu_imagen.jpg')

    # Crear una instancia del segmentador
    segmenter = ImageSegmenter(frame)

    # Aplicar segmentación K-means con k clusters
    segmented_image = segmenter.apply_kmeans_segmentation(k=4)

    # Mostrar la imagen segmentada
    cv2.imshow('Segmented Image', segmented_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Explicación de la Aplicación de la Desfragmentación en este Ejemplo

La desfragmentación en este código permite optimizar la estructura de los datos de la imagen y de esta manera se puede llegar a mejorar el procesamiento, esto ocurre de la siguiente manera:

- Reformateo de la imagen: La imagen se convierte en un array 2D para facilitar su procesamiento. Cada fila del array representa un píxel con sus valores de color.
- Aplicación del algoritmo K-means: Se utiliza el algoritmo K-means para agrupar los píxeles en clusters basados en sus similitudes de color.

Creación de la imagen segmentada: Cada píxel de la imagen original se reemplaza por el color del centro del cluster al que pertenece, creando una imagen segmentada con regiones homogéneas.

La Función ReLU (Rectified Linear Unit)

Definición de la Función ReLU

La función ReLU (Rectified Linear Unit) es una función de activación que es muy utilizada en programas que impliquen el uso de las redes neuronales, esta función se llega a definir matemáticamente de la siguiente manera:

[f(x) = \max(0, x)]

Esto significa que si la entrada (x) es mayor que cero, la salida es (x); de lo contrario, la salida es cero, gracias a esto podemos notar que la simplicidad de esta función la hace muy eficiente y efectiva para

introducir no linealidades en los modelos de redes neuronales.

Funcionamiento de la Función ReLU

La función ReLU se puede aplicar en la salida de las neuronas en una red neuronal, durante el proceso de entrenamiento, cada neurona calcula una suma ponderada de sus entradas y aplica la función ReLU a esta suma, después, el resultado se puede utilizar como la entrada para la siguiente capa de la red y gracias a este proceso se puede permitir que la red neuronal aprenda y pueda representar relaciones complejas en los datos.

El uso de ReLU nos permite ser beneficiados de la siguiente manera:

- **Eficiencia computacional:** El uso de la función ReLU resulta ser rápida y sencilla de calcular, lo que la convierte en una función muy adecuada para redes neuronales grandes y profundas.
- **Evitación del problema del desvanecimiento del gradiente:** A diferencia de las diferentes funciones de activación sigmoides, la función ReLU nos sirve de gran ayuda para poder mitigar el problema del desvanecimiento del gradiente y permite que las redes neuronales se puedan entrener de una manera más efectiva.

Ejemplo de ReLU: Red Neuronal Convolucional

```
import torch
import torch.nn as nn

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 8)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Definir el dispositivo para la computación (GPU si está disponible, si no, CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# Crear una instancia del modelo y moverlo al dispositivo adecuado
model = SimpleCNN().to(device)
# Cargar los pesos del modelo previamente entrenado
model.load_state_dict(torch.load('cnn_model.pth', map_location=device))
# Establecer el modelo en modo evaluación
```

```
model.eval()
print("Modelo cargado correctamente")
```

Explicación de la Aplicación de la Función ReLU en este Ejemplo

La función ReLU en este código se utiliza para introducir no linealidades en la red neuronal convolucional, permitiendo que la red aprenda y represente relaciones complejas en los datos. Aquí te explico brevemente cómo funciona en cada capa:

- **`Capas convolucionales:`** Se aplican dos capas convolucionales (conv1 y conv2) para extraer características de la imagen y después de cada capa convolucional se puede aplicar la función de activación ReLU (relu) para lograr introducir no linealidades y la capa de pooling (pool) para reducir la dimensión de las características.
- **`Capas fully connected:`** Después de las capas convolucionales, la salida se aplana para que pueda ser utilizada en las capas fully connected (fc1 y fc2) y la función ReLU puede ser aplicada en la primera capa fully connected para introducir no linealidades antes de pasar a la capa final.

Conclusión

En conclusión, la desfragmentación de variables y la función ReLU son componentes muy esenciales que pueden ser aplicados en el procesamiento de datos y el aprendizaje automático. La desfragmentación se encarga de optimizar el uso de la memoria y mejora la eficiencia en la gestión de datos, mientras que la función ReLU es crucial para que las redes neuronales aprendan y representen patrones complejos en los datos. Por último, se podría decir que estas técnicas contribuyen significativamente al desarrollo de soluciones avanzadas y eficientes en inteligencia artificial.

Fuentes de Información

- Estructuras de datos en Python
- Función de activación ReLU
- Por qué usamos ReLU en redes neuronales y cómo lo usamos