Image Edge Detection Using Convolutional Neural Network (CNN) Layers with TensorFlow

1. Title

Implementation of Image Edge Detection Using Convolutional Neural Network (CNN) Layers with TensorFlow

---

2. Abstract

Edge detection is a fundamental operation in image processing and computer vision. It helps in identifying object boundaries by detecting sharp changes in pixel intensity. This project demonstrates the implementation of image edge detection using basic Convolutional Neural Network (CNN) layers such as convolution, activation (ReLU), and pooling, implemented using the TensorFlow library. A grayscale image is processed through a convolution kernel to extract edges, followed by non-linearity and down-sampling to enhance important features. The project provides a simple yet effective understanding of how CNN layers work internally for feature extraction.

---

3. Introduction

Image processing plays a vital role in various applications such as medical imaging, object detection, facial recognition, and autonomous vehicles. Edge detection is one of the earliest and most important steps in image analysis. Traditional edge detection techniques include Sobel, Prewitt, and Laplacian operators. In this project, a CNN-based approach is used to perform edge detection using TensorFlow, which mimics how convolutional layers operate in deep learning models.

---

4. Problem Statement

To implement image edge detection using Convolutional Neural Network (CNN) layers such as convolution, activation, and pooling using TensorFlow.

---

5. Objectives

- To understand the working of CNN layers
- To implement convolution operation using TensorFlow
- To apply ReLU activation for non-linearity
- To perform max pooling for feature reduction

- To visualize intermediate outputs of CNN layers

---

6. Tools and Technologies Used

- **Programming Language:** Python
- **Libraries:** TensorFlow, NumPy, Matplotlib
- **Platform:** Windows
- **IDE:** Jupyter Notebook / VS Code

---

7. Methodology

7.1 Image Loading and Preprocessing

- The input image is read using TensorFlow I/O functions
- Converted to grayscale
- Resized to 300×300 pixels
- Normalized to floating-point values

7.2 Convolution Layer

- A 3×3 edge detection kernel is defined
- Convolution is applied using `tf.nn.conv2d`
- This step highlights edges in the image

7.3 Activation Layer (ReLU)

- Rectified Linear Unit (ReLU) is applied
- Removes negative values
- Enhances edge features

7.4 Pooling Layer

- Max pooling with 2×2 window
- Reduces image size
- Retains important edge features

---

8. System Architecture

Input Image → Convolution Layer → ReLU Activation → Max Pooling → Output Image

---

9. Implementation Details

The project uses TensorFlow's low-level neural network functions to simulate CNN layers. The convolution kernel used is similar to a Laplacian filter, which is effective in detecting edges. Matplotlib is used to visualize the original image and outputs after each layer.

Example:

Applying CNN to an Image Let's consider an image and apply the convolution layer, activation layer, and pooling layer operation to extract the inside feature

Input image:

```
#Implement image edge detection using Convolution network layer (CNN) layers with Tensorflow.

import numpy

import tensorflow as tf

import matplotlib.pyplot as plt

from itertools import product

#set the param

plt.rc('figure',autolayout=True)

plt.rc('image',cmap='magma')

#define the kernel
```

```
kernel = tf.constant([[-1,-1,-1],[-1,8,-1],[-1,-1,-1],])


#loading image

image=tf.io.read_file('purple.jpeg')

image=tf.io.decode_jpeg(image, channels=1)

image=tf.image.resize(image, size=[300,300])


#plot the image

img = tf.squeeze(image).numpy()

plt.figure(figsize=(5,5))

plt.imshow(img,cmap='grey')

plt.axis('off')

plt.title('original gray scale image')

plt.show();




#reformat_

image = tf.image.convert_image_dtype(image,dtype=tf.float32)

image= tf.expand_dims(image, axis=0)

kernel= tf.reshape(kernel,[*kernel.shape,1,1])

kernel=tf.cast(kernel,dtype=tf.float32)


#convolution layer

conv_fn=tf.nn.conv2d

image_filter = conv_fn(input=image, filters=kernel,strides=1,padding='SAME',)
```

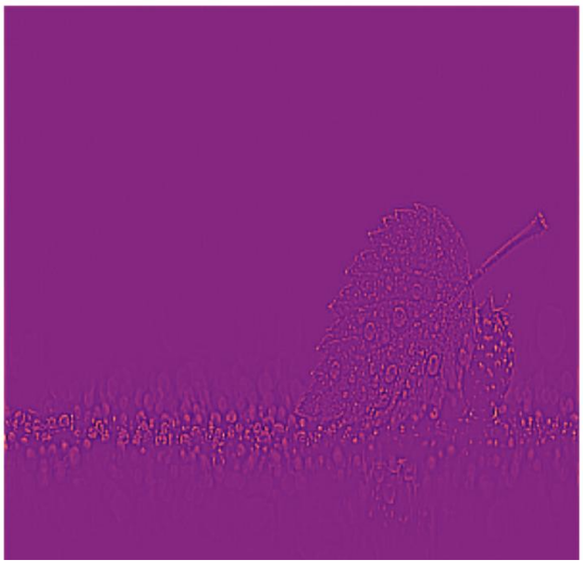
```
plt.figure(figsize=(15,5))


#plot the convolved image

plt.subplot(1,3,1)

plt.imshow(tf.squeeze(image_filter))

plt.axis('off')

plt.title('Convolution')


#activation layer

relu_fn=tf.nn.relu


#image detection

image_detect=relu_fn(image_filter)

plt.subplot(1,3,2)

plt.imshow(tf.squeeze(image_detect))

plt.axis('off')

plt.title('Activation')


#pooling layer

image_condense=tf.nn.pool(

   input=image_detect,

   window_shape=(2,2),

   pooling_type='MAX',

   strides=(2,2),

   padding='SAME')
```

```
#display all results


plt.figure(figsize=(15,5))

plt.subplot(1,3,1)

plt.imshow(tf.squeeze(image_filter),cmap='grey')

plt.title('convolution output')

plt.axis('off')


plt.subplot(1,3,2)

plt.imshow(tf.squeeze(image_detect),cmap='grey')

plt.title('ReLU activation')

plt.axis('off')


plt.subplot(1,3,3)

plt.imshow(tf.squeeze(image_condense),cmap='grey')

plt.title('ReLU activation')

plt.axis('off')


plt.show()
```

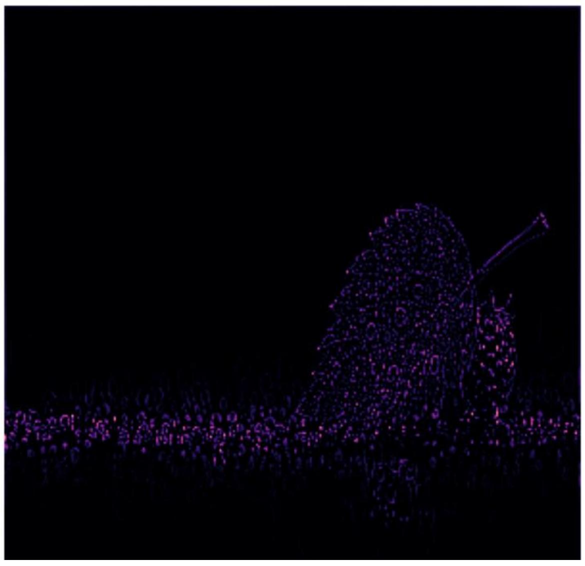
Output:

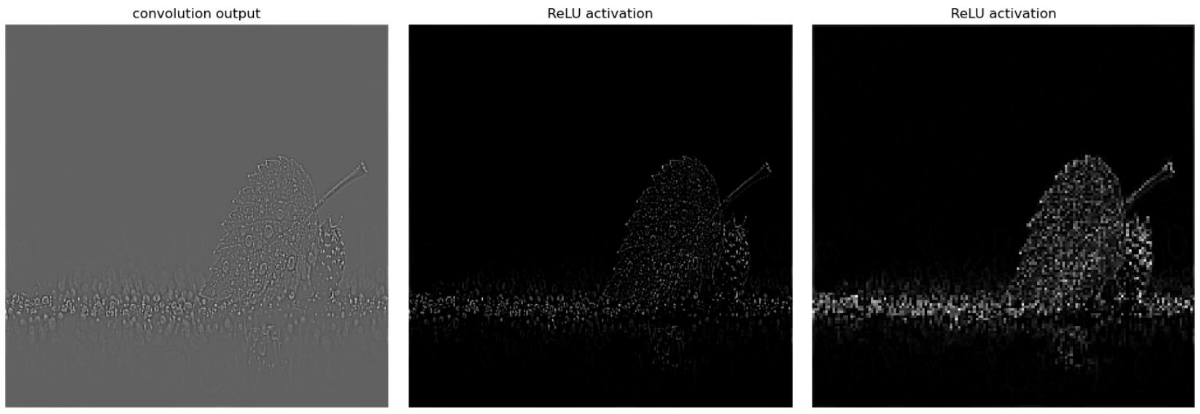
original gray scale image

Convolution



Activation





10. Results and Output

- Original grayscale image is displayed
- Convolution output highlights edges
- ReLU activation removes noise
- Pooling output produces condensed edge features

The outputs clearly demonstrate how CNN layers extract and refine image features step-by-step.

11. Advantages

- Simple and easy to understand
- Demonstrates core CNN concepts
- Uses modern deep learning framework
- Useful for beginners in computer vision

12. Limitations

- Uses a fixed kernel
- Not trained on data
- Works on a single image