1. Project Title

Ganesh Image Feature Extraction using Convolutional Neural Network (CNN) Operations

---

2. Overview

This project demonstrates the application of **Convolutional Neural Network (CNN) operations** on a grayscale image of Lord Ganesh. It shows how CNN layers—**convolution, activation, and pooling**—can extract features such as edges and important patterns from an image. The project uses **TensorFlow** for computation and **Matplotlib** for visualization.

The project focuses on **feature extraction**, which is a fundamental step in many computer vision tasks like object recognition, image classification, and detection.

---

3. Objective of the Project

- To apply convolutional operations on an image to detect edges and important features.
- To understand the role of **ReLU activation** in highlighting positive feature responses.
- To demonstrate **max pooling** for condensing image features and reducing dimensions.
- To provide visual outputs for each CNN stage for better understanding.

---

4. Problem Statement

Feature extraction from images is crucial for many computer vision applications. Manually identifying edges or patterns is time-consuming and error-prone. Using CNN operations automates this process, allowing computers to learn and highlight important patterns efficiently. This project implements a simple CNN pipeline to extract edges and features from a Ganesh image.

---

5. Dataset

- Single image: Ganesh.jpg
- Preprocessing:
    - Resized to **300×300 pixels**
    - Converted to **grayscale**

- Normalized to **[0, 1]** range for CNN processing

For future work, a dataset of multiple Ganesh images could be used for deeper CNN training and classification.

---

6. Methodology

The project follows a **step-by-step CNN feature extraction pipeline**:

1. **Define Convolution Kernel**:
   - A **3×3 edge-detection kernel** is used to highlight edges in the image.
2. [-1 -1 -1]
3. [-1 8 -1]
4. [-1 -1 -1]
5. **Preprocess Image**:
   - Read the image using tf.io.read_file and tf.image.decode_jpeg.
   - Resize to 300×300 pixels and convert to grayscale.
   - Normalize pixel values to float32 in [0,1].
   - Expand dimensions for CNN input shape: [batch, height, width, channels].
6. **Apply Convolution**:
   - Perform convolution with stride 1 and padding 'SAME'.
   - Output highlights **edges** and contours in the image.
7. **Activation (ReLU)**:
   - Apply **ReLU (Rectified Linear Unit)** to keep only positive values.
   - Negative pixel values are set to zero, enhancing prominent features.
8. **Max Pooling**:
   - Apply **2×2 max pooling** to reduce spatial dimensions.
   - Retains strongest features while condensing the image.
9. **Visualization**:
   - Use Matplotlib to display results at each stage:
     - Original grayscale image

        - After convolution
        - After ReLU activation
        - After max pooling

---

7. Training Setup

This project focuses on **feature extraction**, not training a CNN model.

- No training is performed; operations are applied directly on the image.

---

8. Optimizer

- Not applicable (no model training).

---

9. Evaluation Metrics

- Visual inspection of feature extraction results:
    - **Convolution**: Check for proper edge detection.
    - **ReLU**: Ensure negative values removed, features highlighted.
    - **Pooling**: Verify image dimension reduction and feature retention.

---

10. Tools

- **Programming Language**: Python 3.x
- **Libraries**:
    - TensorFlow for convolution, activation, and pooling
    - Matplotlib for visualization
    - NumPy for numerical operations (optional)

---

11. Implementation

- Code implements a **simple CNN pipeline**:
    1. Load and preprocess Ganesh image.
    2. Apply 3×3 convolution to detect edges.

3. Apply ReLU activation.
4. Apply 2×2 max pooling.
5. Visualize results side by side for analysis.

- The process mimics a basic **CNN layer pipeline** in real-world computer vision applications.

---

12. Results

The following outputs are generated:

Stage	Description
Original Gray	300×300 grayscale Ganesh image.
Convolution (Edges)	Edge-detected image highlighting contours and boundaries.
Activation (ReLU)	Only positive edges highlighted, negative values removed.
Pooling (Condensed)	Reduced-size image (150×150) retaining dominant features.

All outputs are visualized in a 1×4 subplot using Matplotlib.

---

13. Challenges

- Working with a **single image** limits the generalizability of the feature extraction pipeline.
- Choosing appropriate **kernel values** is crucial; wrong values can produce noisy or unclear edges.
- Understanding the **effects of convolution, activation, and pooling** requires careful observation.
- Visual clarity depends on the **contrast and quality of the original image**.

```
import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt
```

```
# Set parameters for visualization
plt.rc('figure', autolayout=True)
plt.rc('image', cmap='magma')

# 1. Define the kernel (Filter)
# Must be reshaped to [filter_height, filter_width, in_channels, out_channels]
kernel = tf.constant([[-1, -1, -1],
                      [-1,  8, -1],
                      [-1, -1, -1]], dtype=tf.float32)
kernel = tf.reshape(kernel, [3, 3, 1, 1])

# 2. Load and Preprocess the image
# Using tf.io.read_file and tf.image.decode_jpeg for raw processing
image_raw = tf.io.read_file('Ganesh.jpg')
image = tf.io.decode_jpeg(image_raw, channels=1)
image = tf.image.resize(image, size=[300, 300])

# Reformat image for convolution: [batch, height, width, channels]
image = tf.image.convert_image_dtype(image, dtype=tf.float32)
image = tf.expand_dims(image, axis=0)

# 3. Convolution Layer (Filtering)
# tf.nn.conv2d requires float32 and specific 4D shapes
image_filter = tf.nn.conv2d(
    input=image,
    filters=kernel,
    strides=[1, 1, 1, 1], # 2026 standards often use 4-element lists for explicit strides
    padding='SAME',
```

```
)

# 4. Activation Layer (Detecting)
# ReLU removes negative values to highlight detected features
image_detect = tf.nn.relu(image_filter)

# 5. Pooling Layer (Condensing)
# tf.nn.pool is a general N-D pooling op; window_shape and strides are spatial (H, W)
image_condense = tf.nn.pool(
    input=image_detect,
    window_shape=(2, 2),
    pooling_type='MAX',
    strides=(2, 2),
    padding='SAME',
)

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

# Original (Grayscale)
plt.subplot(1, 4, 1)
plt.imshow(tf.squeeze(image), cmap='gray')
plt.title('Original Gray')
plt.axis('off')

# After Convolution
plt.subplot(1, 4, 2)
plt.imshow(tf.squeeze(image_filter))
```

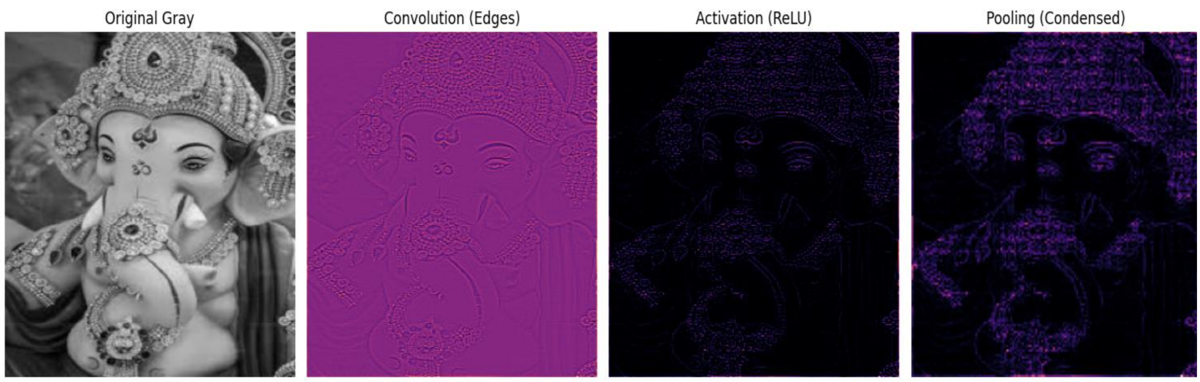
```
plt.title('Convolution (Edges)')
plt.axis('off')

# After ReLU
plt.subplot(1, 4, 3)
plt.imshow(tf.squeeze(image_detect))
plt.title('Activation (ReLU)')
plt.axis('off')

# After Max Pooling
plt.subplot(1, 4, 4)
plt.imshow(tf.squeeze(image_condense))
plt.title('Pooling (Condensed)')
plt.axis('off')

plt.show()
```

14. Output:



15. Conclusion

This project successfully demonstrates how **CNN operations can extract meaningful features** from an image:

- Convolution highlights edges.
- ReLU emphasizes positive feature responses.
- Max pooling condenses features and reduces image size.

This forms the foundation for **advanced computer vision tasks**, such as image classification or object detection.