Introduction :

Human Activity Recognition (HAR) using Convolutional Neural Networks (CNNs) is a common application of deep learning, particularly in the field of wearable technology and motion analysis. Here's a general outline of how you can apply a CNN for HAR:

1. **Data collection and Preprocessing :**

- Collect a dataset of sensor readings (accelerometer, gyroscope, etc.) from wearable devices or other sources. Each data point should represent a snapshot of sensor readings at a particular time.
- Preprocess the data, which may include normalization, filtering, and segmentation into fixed-length time windows.

2. **Data Representation :**

- Represent each time window of sensor readings as an input tensor suitable for feeding into a CNN. This tensor could have dimensions like (time_steps, num_features, 1), where time_steps represents the number of readings in each window, and num_features represents the number of sensor channels.

3. **Designing the CNN Architecture :**

- Design a CNN architecture suitable for processing the input data. This typically involves stacking convolutional layers followed by pooling layers to extract spatial features from the sensor readings.
- You may also include additional layers like dropout and batch normalization to improve generalization and training stability.
- Flatten the output of the convolutional layers to feed it into fully connected layers for classification.

4. **Training The CNN :**

- Split the dataset into training, validation, and testing sets.
- Train the CNN using the training set. During training, the network learns to classify the activities based on the sensor readings.
- Use techniques like cross-validation and hyperparameter tuning to optimize the model's performance.

5. **Evaluation and Testing :**

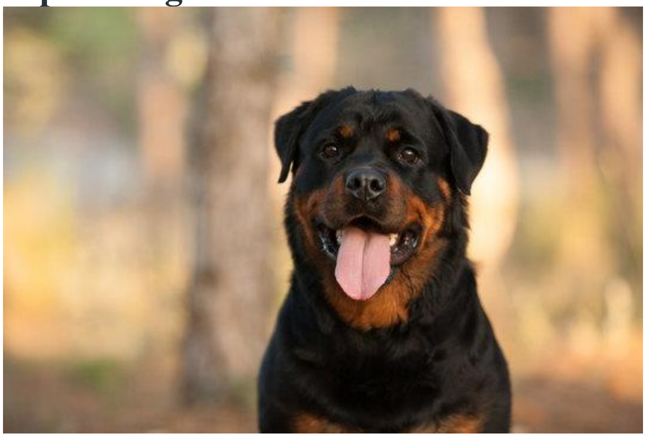
- Evaluate the trained model using the validation set to tune hyperparameters and monitor for overfitting.
- Test the final model using the testing set to assess its performance on unseen data.
- Calculate metrics such as accuracy, precision, recall, and F1-score to quantify the model's performance.

6. **Deployment :**

- Once satisfied with the model's performance, deploy it for real-world applications. This could involve integrating it into a mobile app or other systems for real-time activity recognition.

Example: Let’s consider an image and apply the convolution layer, activation layer, and pooling layer operation to extract the inside feature.

Input image:



Steps :

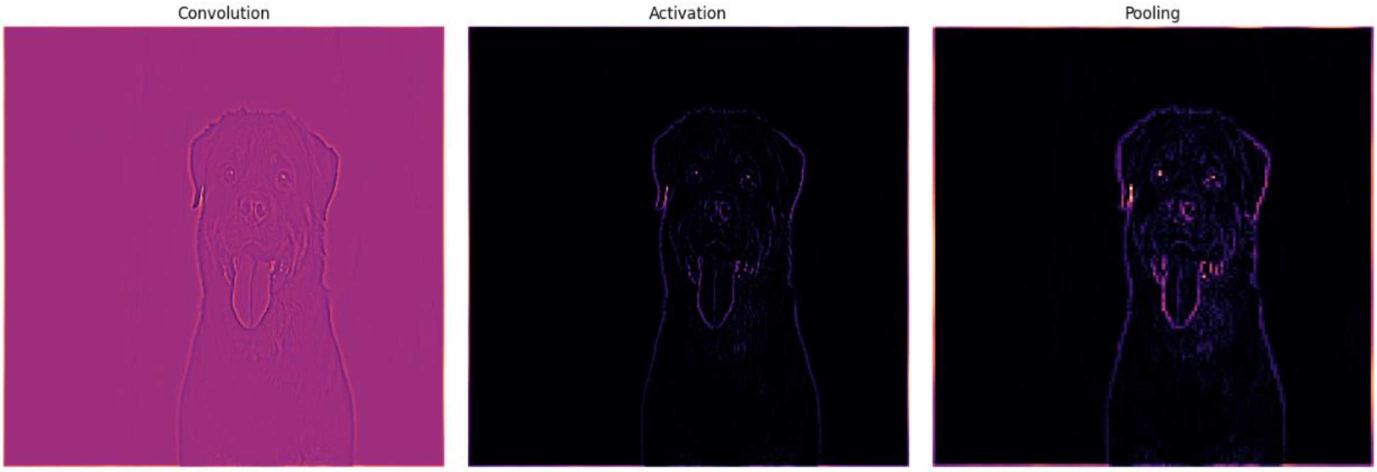
- import the necessary libraries
- set the parameter
- define the kernel
- Load the image and plot it.
- Reformat the image
- Apply convolution layer operation and plot the output image.
- Apply activation layer operation and plot the output image.
- Apply pooling layer operation and plot the output image.

Program :

```
# import the necessary libraries
import numpy as np
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],
                     ])

# load the image
image = tf.io.read_file(dog.jpg')
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='gray')
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, # or (1, 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(
        # Reformat for plotting
        tf.squeeze(image_detect)
)

plt.axis('off')
plt.title('Activation')

# Pooling layer
pool = tf.nn.pool
image_condense = pool(input=image_detect,
                      window_shape=(2, 2),
                      pooling_type='MAX',
                      strides=(2, 2),
                      padding='SAME',
                      )

plt.subplot(1, 3, 3)
plt.imshow(tf.squeeze(image_condense))
plt.axis('off')
plt.title('Pooling')
plt.show()
```

Output :


Convolution
Activation
Pooling