

# How Filter Sizes and Strides Shape Feature Detection in CNNs

Roshni Krishna

November 28, 2024

# 1 Introduction

Convolutional Neural Networks (CNNs) are a specialized type of deep learning model created for analyzing images, but they can also be used for other types of data with grid-like formats (like audio spectrograms). CNNs are now crucial in different sectors, such as computer vision, employed for activities such as image classification, object detection, and face recognition. This guide provides an introduction to CNNs in machine learning or deep learning, covering the basics, building a simple CNN model, and visualizing learned features to comprehend its functioning.

The purpose of this report is to present you with an accessible explanation of CNNs and show you how to create and educate a CNN model on CIFAR-10, a popular dataset featuring images from 10 diverse categories. In addition, we will investigate how CNNs visualize images by examining their filters and feature maps.

## Understanding Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a special type of neural network that are well-suited for image processing tasks. Unlike traditional neural networks, which are fully connected, CNNs use layers that apply convolution operations to the input data. This process allows CNNs to automatically learn spatial hierarchies of features, meaning they can detect simple patterns (like edges) in the early layers and more complex patterns (like objects) in deeper layers [3,4].

### Key Components of a CNN

- **Convolutional Layer (Conv Layer):** The core building block of a CNN, where small filters (also called kernels) slide over the image to detect features like edges or textures [1]. This operation preserves spatial relationships by learning feature maps.
- **ReLU Activation Function:** After each convolution operation, a non-linear activation function like ReLU (Rectified Linear Unit) is applied to introduce non-linearity into the network, allowing it to learn more complex patterns [5].
- **Pooling Layer:** A downsampling operation that reduces the spatial dimensions of the image (i.e., the width and height), making the model computationally more efficient and reducing the risk of overfitting [6].
- **Fully Connected Layer:** The final layers of the network, which connect all the features learned by the convolutional and pooling layers to output a classification or prediction [3].

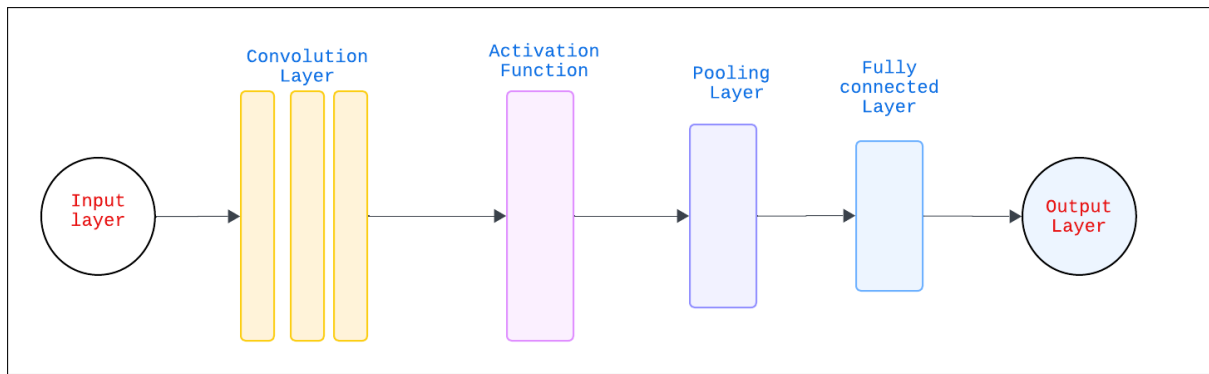


Figure 1: Simple CNN Architecture

## 2 Example :- Applying Convolutional Neural Networks on an Image

Let's analyze an image and utilize the convolution layer, activation layer, and pooling layer operation to extract the internal features.



Figure 2: input image

The sequence of actions in the execution of code is as follows:

1. **Make use of libraries:** Utilize libraries like TensorFlow, NumPy, and Matplotlib for image processing and visualization requirements.
2. **Define the Kernel:** Use a 3x3 Laplacian filter:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

to detect edges in the image.

3. **Submit the image for analysis:** Load the picture (e.g., `dog.jpg`) and prepare it for processing.

#### 4. Preprocessing the image:

- (a) Access and display the image file.
- (b) Transform the image to grayscale and resize it to 300x300 pixels for uniform processing.
- (c) Use Matplotlib to display the grayscale image for visualization.

#### 5. Reorganize and normalize the data:

- (a) Convert image values to `float32` for normalization.
- (b) Prepare the image and kernel to be compatible with TensorFlow operations.

#### 6. Apply Convolution:

- (a) Use the defined filter in a 2D convolutional layer to detect edges by highlighting regions with noticeable intensity changes.
- (b) Implement the *ReLU* (Rectified Linear Unit) activation function to set negative values to zero, retaining only significant features.

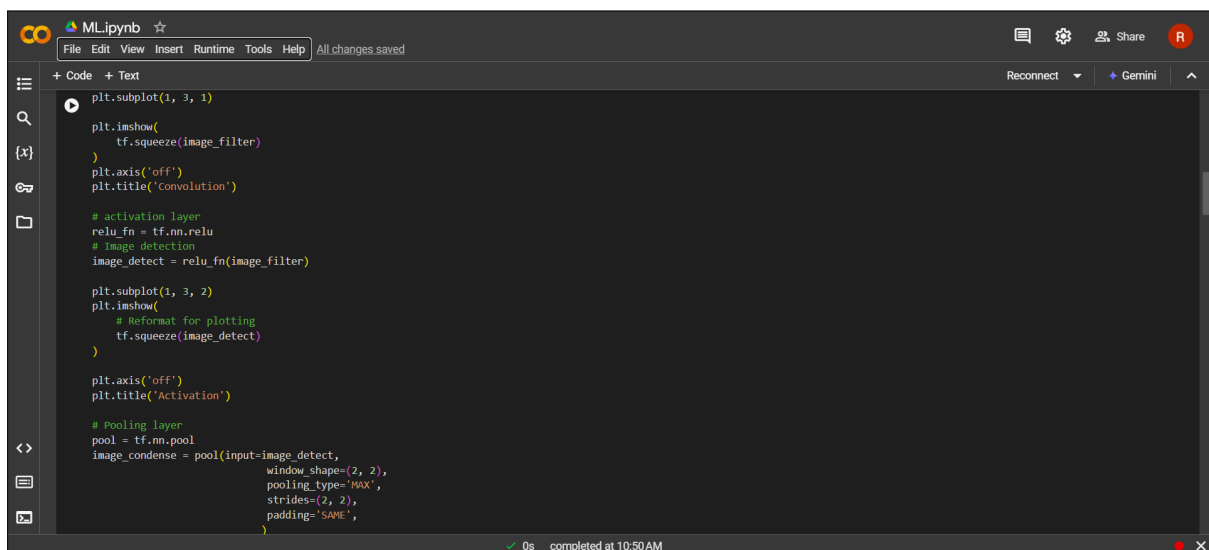
#### 7. Use Pooling:

Apply max pooling with a 2x2 window to reduce the image size and focus on essential features.

#### 8. Show Results:

Display the following stages:

- (a) The edge detection result generated from the convolution operation.
- (b) Enhanced features in the image after activation.
- (c) A summary image showing concise features after pooling.



```
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',
                      )
```

Figure 3: python code

This series of actions demonstrates basic feature extraction techniques in image processing. The output is as follows :

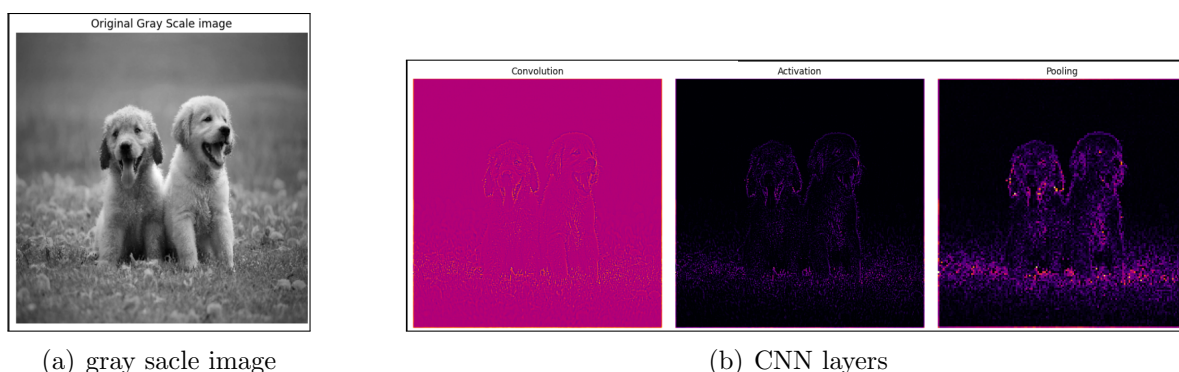


Figure 4: Applying CNN over a image

### 3 CNN Filters and Strides

The purpose of this tutorial is to offer a clear explanation of how filter sizes and strides affect feature detection in CNNs through practical visual aids.

#### 3.1 What is Kernel ?

In CNNs, kernels (or filters) are tiny matrices employed for conducting convolution operations on the input data. These kernels play a crucial role in extracting characteristics from input images or other types of multidimensional data.

#### Kernel Size

The size of the filter, also called the *kernel size*, dictates the spatial extent of the area that the convolutional filter includes while moving across the input image or feature map. It outlines the section that the filter observes and utilizes for feature extraction. A filter is a compact matrix with a set size (such as  $3 \times 3$  or  $5 \times 5$ ) that applies its weights to the pixel values of the input it covers. The total values are added together to generate one outcome, indicating the level of presence of that particular feature in the area.

#### Some Typical Filter Sizes

- **Filters that are  $3 \times 3$  in size, also known as small filters:**
  - Frequently utilized in the majority of CNN designs.
  - Efficient in identifying intricate patterns such as edges, corners, and minor textures.
  - Stacking multiple  $3 \times 3$  filters in layers can simulate the impact of larger filters while introducing non-linearity between the layers.
- **Larger filters, either  $5 \times 5$  or  $7 \times 7$  in size:**
  - Capture wider trends and a larger scope of information in the input.

- Beneficial in cases where the input is mainly characterized by extensive patterns, such as big objects or shapes.
- More costly in terms of computation, due to increased parameters and operations.

## Effects of Filter Size

- **smaller filters:** Concentrate on tiny details such as edges or small textures but need deeper networks to grasp overall patterns.
- **Larger filters:** Condense extensive features but might miss smaller intricacies. When paired with large strides, they also decrease the size of the feature map more quickly.

## 3.2 Strides

In CNNs, stride is the amount of pixels the filter moves across the input image or feature map during convolution.

- Stride of 1: The filter progresses by one pixel per movement, leading to extensive overlap and thorough feature extraction. This results in feature maps that have a size comparable to the input (depending on padding).
  - The filter progresses pixel by pixel without jumping.
  - Generates detailed feature maps of high resolution by incorporating input information from each region to the output.
  - Requires more computations, leading to higher computational expenses.
- Stride of 2 or higher: The filter jumps over pixels while moving, decreasing the overlap and downsampling the feature map. This leads to reduced output feature maps and may enhance computational efficiency while sacrificing spatial information.
  - The filter skips pixels as it moves, leading to less overlap between regions.
  - Generates coarse feature maps with low resolution, focusing on broader patterns and downsampling to reduce input size.
  - Decreases computational cost by conducting fewer calculations.

## 4 Analyzing the Impact of Kernel and Stride Size on Image Processing

There are several distinct effects observed when performing convolution operations on an image with varying kernel sizes and stride values. These variations emphasize the ability of convolutional layers to extract and emphasize different features of an image. The effects of kernel and stride size on image processing are demonstrated on a image below :

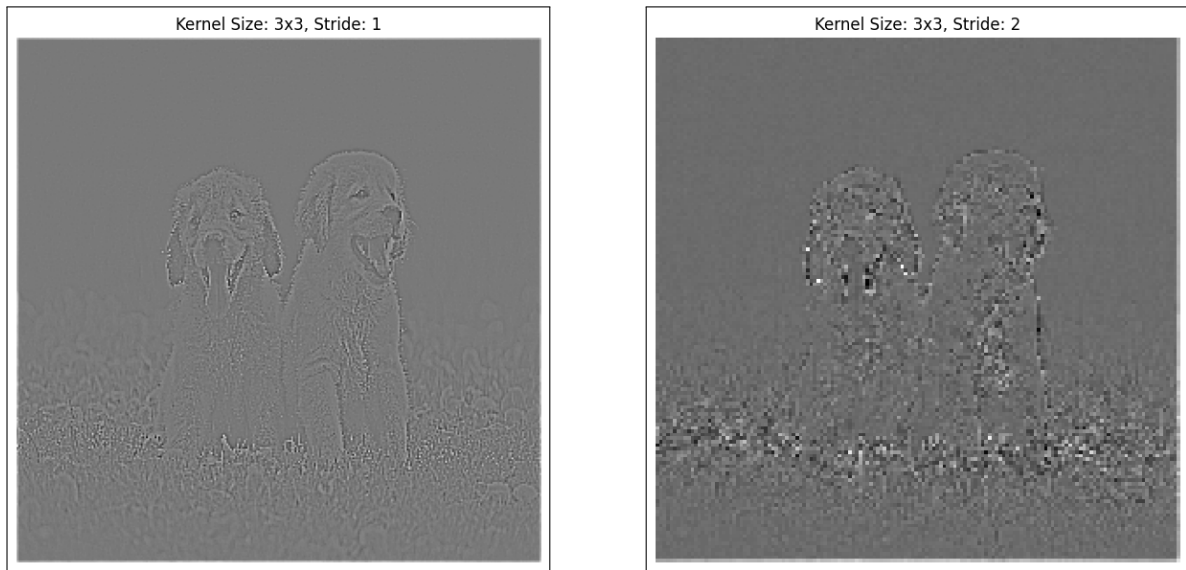


Figure 5: *Kernel size = 3*

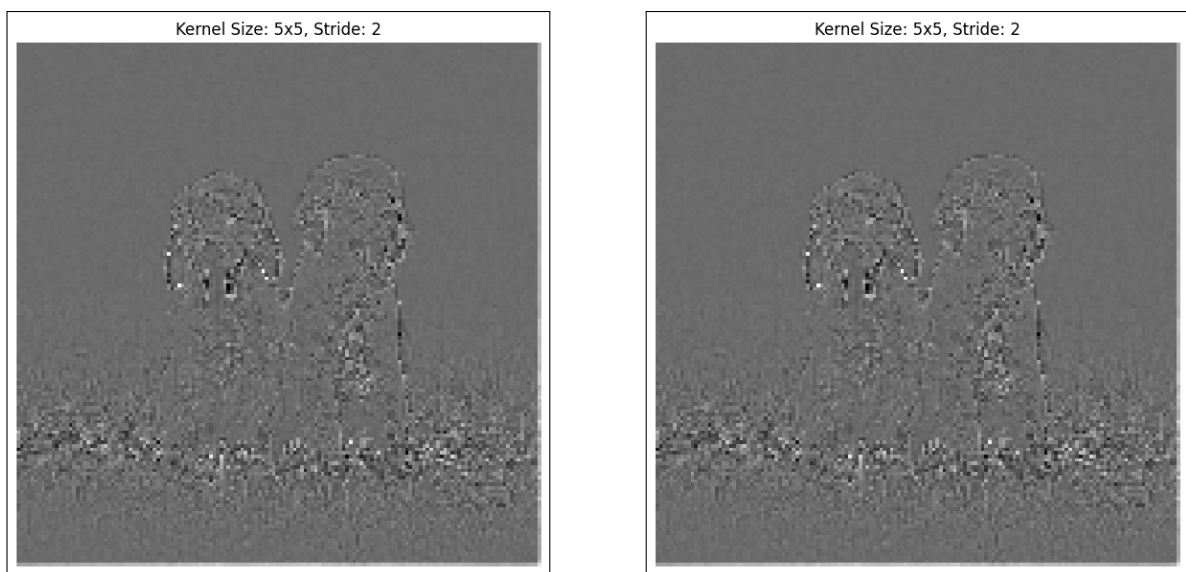
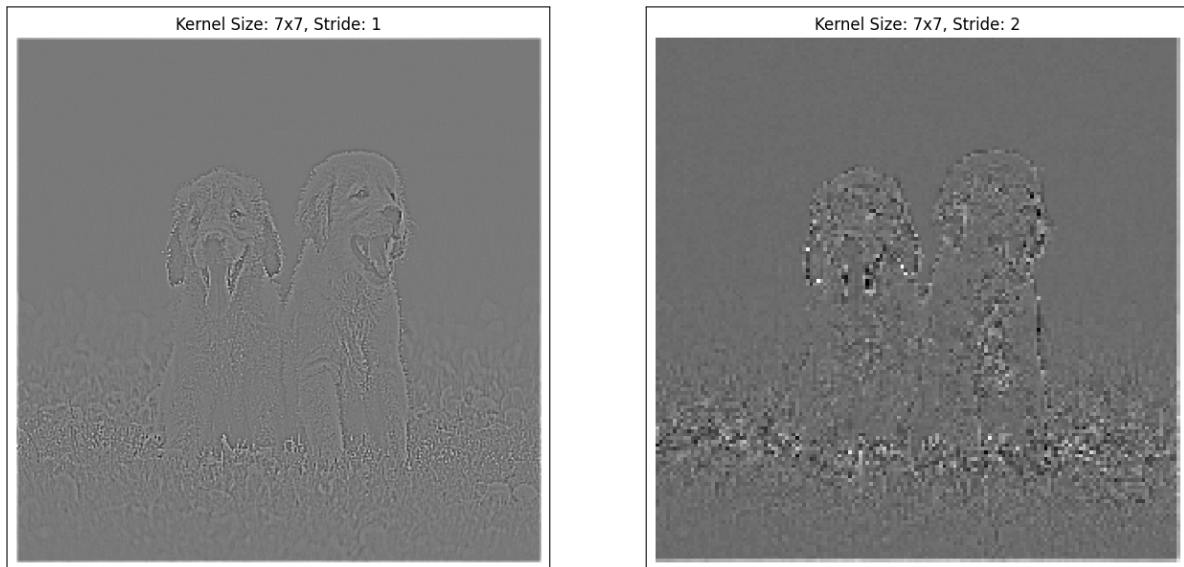


Figure 6: *Kernel size = 5*

Figure 7: *Kernel size = 7*

```

def apply_convolution(image, kernel_size, stride):
    # Create a kernel (simple edge detection kernel)
    kernel = tf.constant([[-1, -1, -1],
                          [-1,  8, -1],
                          [-1, -1, -1]])

    # Pad the kernel to the desired size instead of resizing
    # calculate padding for each dimension
    pad_h = (kernel_size - 3) // 2 # 3 is the original kernel size
    pad_w = (kernel_size - 3) // 2

    # Pad the kernel using tf.pad
    kernel = tf.pad(kernel, [[pad_h, pad_h], [pad_w, pad_w]], mode='CONSTANT')

    kernel = tf.reshape(kernel, [kernel_size, kernel_size, 1, 1])
    kernel = tf.cast(kernel, dtype=tf.float32)

    # Apply convolution
    conv_fn = tf.nn.conv1d
    convolved_image = conv_fn(
        input=image,
        filters=kernel,
        strides=(stride, stride),
        padding='SAME',
    )

    # Visualize the output
    plt.figure(figsize=(6, 6))
    plt.imshow(tf.squeeze(convolved_image).numpy(), cmap='gray')
    plt.axis('off')
    plt.title(f"Kernel Size: {kernel_size}x{kernel_size}, Stride: {stride}")
  
```

Figure 8: input image



Table 1: Effect of Kernel Size and Stride in CNNs

Aspect	Description
<b>Effect of Kernel Size</b>	
<b>Small Kernel (e.g., <math>3 \times 3</math>)</b>	<ul style="list-style-type: none"> <li>• Focuses on fine details, like edges and textures.</li> <li>• Produces sharper outputs with precise feature detection.</li> <li>• Useful for tasks like edge detection and identifying small patterns.</li> </ul>
<b>Large Kernel (e.g., <math>7 \times 7</math>)</b>	<ul style="list-style-type: none"> <li>• Covers a larger area in the image.</li> <li>• Extracts broader, more generalized patterns while smoothing out finer details.</li> <li>• Ideal for identifying larger objects or regions in the image.</li> </ul>
<b>Effect of Stride</b>	
<b>Small Stride (e.g., 1)</b>	<ul style="list-style-type: none"> <li>• Filters overlap extensively, retaining detailed spatial information.</li> <li>• Results in a high-resolution output with more nuanced features.</li> <li>• Computationally more expensive due to the greater number of calculations.</li> </ul>
<b>Large Stride (e.g., 2)</b>	<ul style="list-style-type: none"> <li>• Reduces overlap, leading to coarser outputs by downsampling the image.</li> <li>• Simplifies the feature map, making it computationally efficient.</li> <li>• Highlights broader patterns at the cost of finer details.</li> </ul>

## 5 Applications in the Real World

Selecting the appropriate kernel size and stride is essential in order to meet the specific objectives of an application.

### 5.1 Reduced Kernel and Stride Sizes:

**Imaging in the medical field:** Analyzing X-rays, MRIs, or CT scans for tumors or abnormalities demands high spatial resolution to avoid overlooking important information. Smaller kernels and strides can detect slight textural variances that suggest abnormalities.

**Facial recognition technology:** Precisely recognizing facial features such as the eyes, nose, and mouth requires maintaining intricate details, a task at which smaller kernels and strides excel.

### 5.2 Bigger Kernel Sizes and Bigger Stride Lengths:

**General Image Classification:** Efficiency is prioritized over extreme detail when it comes to recognizing objects in large datasets such as ImageNet. Bigger kernel sizes and large strides can capture basic, indicative patterns to enhance computational efficiency with minimal impact on accuracy.

**Analysis of the scene:** Larger kernels and strides are effective in capturing large-scale features necessary for identifying broader patterns like buildings, roads, or vehicles in tasks such as satellite imagery analysis or autonomous driving.

## 6 Interactive Visualization of CNNs

For a greater comprehension of how Convolutional Neural Networks (CNNs) analyze images, experiment with the DeepLizard Interactive CNN Visualization Tool. This tool is designed to make abstract concepts such as convolution, activation functions, and pooling layers more concrete and understandable.

1. Click on this link to access the tool: [DeepLizard Interactive CNN Tool](#).
2. Try different combinations of kernel size and stride, to see their impact on the results.
3. Observe how altering these parameters affects the feature maps produced by the network.

This interactive tool is an excellent resource for gaining a deeper understanding of how CNNs process and extract information from images.

**Take the opportunity to engage with this tool!**

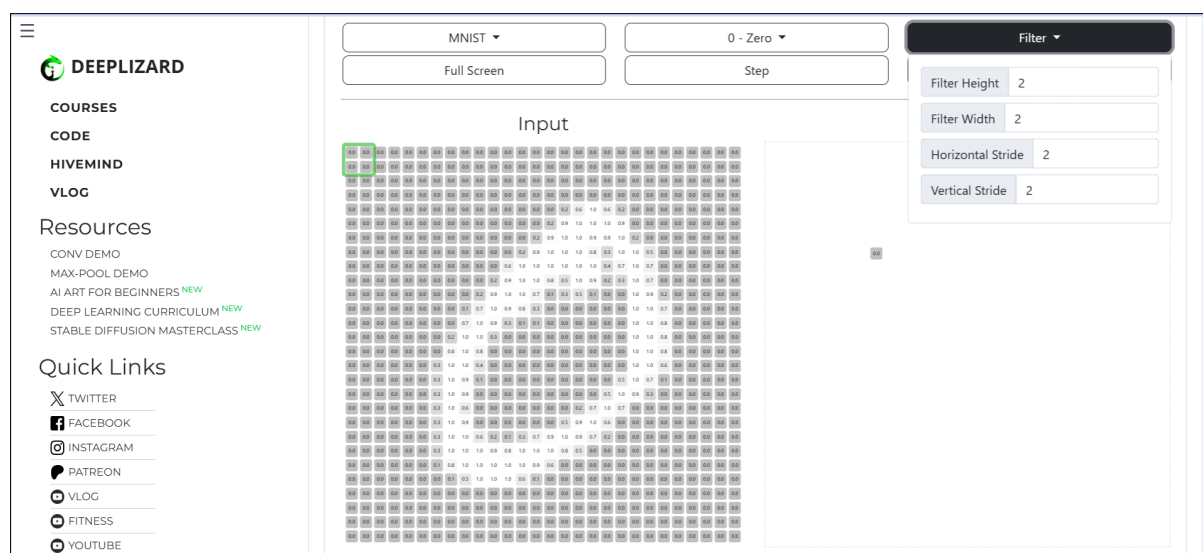


Figure 9: A snippet from DeepLizard Interactive CNN Tool

## 7 Deeper Insights on Kernel Size and Strides

- Overlapped Features and Overfitting (Small Stride)

Heavy overlap between adjacent convolution operations is caused by using a small stride (e.g., 1). High-resolution feature maps remain, but it also introduces redundant information in learned attributes.

- Example:

In semantic segmentation, small strides are often used initially to maintain spatial detail, but later layers may increase strides to reduce feature redundancy and computational complexity.

- Computational Cost Versus Precision

The size and stride of the kernel directly affect:

Aspect	Effect
<b>Accuracy</b>	<ul style="list-style-type: none"> <li>Small kernels and strides capture more details but require higher computational resources.</li> <li>Large kernels and strides sacrifice detail but improve computational efficiency.</li> </ul>
<b>Computational Cost</b>	<ul style="list-style-type: none"> <li>Large strides reduce the size of feature maps faster, lowering memory and computation needs.</li> <li>Example: A stride of 2 effectively halves the dimensions of the output feature map.</li> </ul>

### Research-Backed Insights

- A study on ResNet showed that using a small  $3 \times 3$  kernel in most layers balances computational cost and accuracy due to its flexibility in detecting fine and coarse features. [2]

- Models like YOLO (You Only Look Once) employ large strides in earlier layers to quickly downsample images, achieving faster inference times while focusing on critical object-level features. [7]

**GitHub Repository:** <https://github.com/rosh1728/MachineLearningCNN>

## References

- [1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.
- [2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 25, pages 1097–1105, 2012.
- [4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [5] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.
- [6] Marc’Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1185–1192, 2007.
- [7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.