

Lab 5.1

Convolution-kernel-&-Feature-map

Lab5.1 : CNN Feature maps

```
[1]: import numpy as np
import cv2
import matplotlib.pyplot as plt
import torch
from torch import nn
import torchvision
import torchvision.models as models
import torchvision.transforms as transforms
```

Import Library ที่จำเป็นที่ต้องใช้ใน Lab 5.1 นี้

```
[2]: ### START CODE HERE ###
# vgg16 = models.vgg16(pretrained=True)
vgg16 = models.vgg16(weights='DEFAULT')
vgg16.eval() # Set the model to evaluation mode
print(vgg16)
### END CODE HERE ###

VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Explore all layers in feature extractor part. [Read more.](#)

[2] `vgg16 = models.vgg16(weights='DEFAULT')`

- loads the VGG16 model with default pre-trained weights

`vgg16.eval()`

- Set the model to evaluation mode -> tells the model that you're about to use it for inference (making predictions on new data), not for training.

```
[3]: ### START CODE HERE ###

# Extract the feature extractor part
features = vgg16.features

# Print the layers in the feature extractor part
print("Layers in the feature extractor part of VGG16:")
for i, layer in enumerate(features.children()):
    print(f"Layer {i}: {layer}")

# print("\nSequential structure of the feature extractor part:")
# print(features)

### END CODE HERE ###

Layers in the feature extractor part of VGG16:
-----
Layer 0: Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
-----
Layer 1: ReLU(inplace=True)
-----
Layer 2: Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
-----
Layer 3: ReLU(inplace=True)
-----
Layer 4: MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
-----
Layer 5: Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
-----
Layer 6: ReLU(inplace=True)
-----
Layer 7: Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
-----
Layer 8: ReLU(inplace=True)
-----
Layer 9: MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
-----
Layer 10: Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
-----
Layer 11: ReLU(inplace=True)
-----
Layer 12: Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
-----
Layer 13: ReLU(inplace=True)
-----
Layer 14: Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
-----
Layer 15: ReLU(inplace=True)
-----
Layer 16: MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
-----
Layer 17: Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
-----
```

[3] features = vgg16.features

- Extract the feature extractor part of the VGG16 model
- for i, layer in enumerate(features.children())
- loop all the layers (or "children") within the feature extractor part of the VGG16 model.

- i: ตัวแปรสำหรับเก็บหมายเลขของ layer
- layer: ตัวแปรสำหรับเก็บข้อมูลเกี่ยวกับ layer ปัจจุบัน

และแสดงผลออกมาในแต่ละ layer

```
[4]: ### START CODE HERE ###
# Access the first Convolutional Layer and ReLU Layer
first_conv = features[0] # Conv2d Layer
first_relu = features[1] # ReLU Layer

# Print details for the first Convolutional Layer
print("First Convolutional Layer (Conv2d):")
print("Weights shape:", first_conv.weight.shape)
# Iterate through each kernel
for kernel_index in range(first_conv.weight.shape[0]): # first_conv.weight.shape[0] is the number of kernels (output channels)
    print(f"\nKernel : {kernel_index}")
    print("*****")

    # Iterate over the input channels of the current kernel
    for channel_index in range(first_conv.weight.shape[1]): # first_conv.weight.shape[1] is the number of input channels
        print(f"Channel : {channel_index}")
        kernel_weights = first_conv.weight[kernel_index, channel_index].detach().numpy() # Get the weights of the ith channel of the current kernel
        print(kernel_weights)
        print("Min coefficient:", kernel_weights.min())
        print("-----")

# Print details for the ReLU Layer
print("\nFirst ReLU Layer:")
# print("Weights shape:", first_relu.weight.shape)
print("Type:", type(first_relu))
print("ReLU Layer (No weights or biases to display)")
print("*****")

### END CODE HERE ###

First Convolutional Layer (Conv2d):
Weights shape: torch.Size([64, 3, 3, 3])

Kernel : 0
*****
Channel : 0
[[[-0.5537306  0.1427047  0.5289615 ]
  [-0.58312404  0.35655147  0.76566225]
  [-0.69022113 -0.04801885  0.48409155]]]
Min coefficient: -0.69022113
-----
Channel : 1
[[[ 0.17548391  0.00986297 -0.08141315]
  [ 0.04408892 -0.07032251 -0.26035076]
  [ 0.13239175 -0.1727862 -0.1322633 ]]]
Min coefficient: -0.26035076
-----
Channel : 2
```

[4] **first_conv = features[0]**

- accesses the first layer in the feature extractor part of the VGG16 model, which is a Conv2d (convolutional) layer.

first_relu = features[1]

- accesses the second layer, which is a ReLU (Rectified Linear Unit) activation function.
- วนลูปผ่านแต่ละเคอร์เนล (kernel) โดยที่:
 - first_conv.weight.shape[0] คือจำนวนเคอร์เนล (ช่องสัญญาณขาออก)
 - first_conv.weight.shape[1] คือจำนวนช่องสัญญาณขาเข้า
- ดึงน้ำหนักของ kernel:
 - kernel_weights = first_conv.weight[kernel_index, channel_index].detach().numpy() ดึงน้ำหนักของช่องสัญญาณปัจจุบันของเคอร์เนลปัจจุบัน และแปลงเป็น numpy array
- Prints the min value in the kernel's weights, which gives insight into the range of values that the kernel has learned.

นั่นคือโค้ดนี้จะเข้าถึง layer: Convolutional และ ReLU แรกจากโมเดล แล้วแสดงรายละเอียดของ weights ของ layer: Convolutional แรก โดยวนลูปผ่านแต่ละคอร์เนลและช่องสัญญาณขาเข้าเพื่อแสดงน้ำหนักของแต่ละช่องสัญญาณ

- โดยที่ Prints the type of the ReLU layer to confirm it as a torch.nn.ReLU object, which does not have weights or biases.

```
[6]: ### START CODE HERE ###
print("Bias :", first_conv.bias)
#print("Bias :", first_relu.bias) # 'ReLU' object has no attribute 'bias'
### END CODE HERE ###

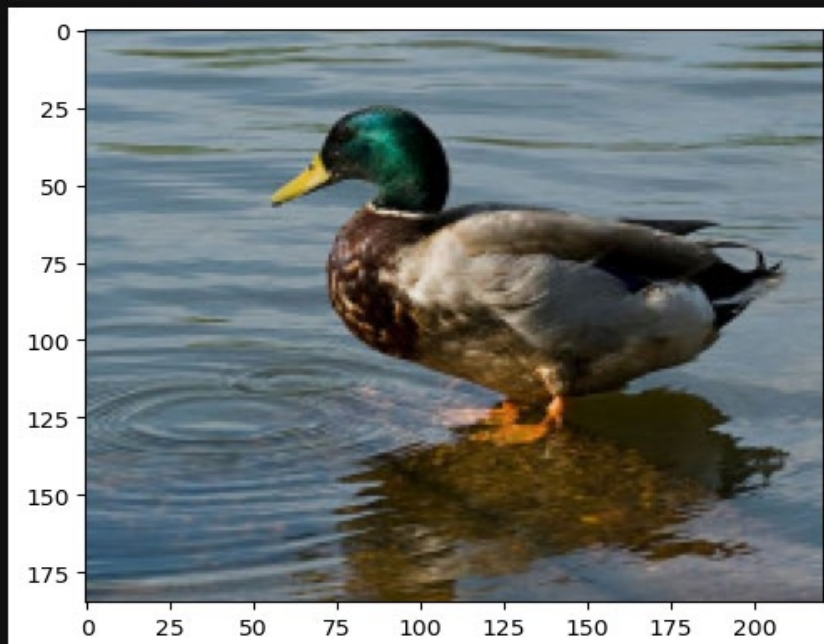
Bias : Parameter containing:
tensor([ 0.4034,  0.3778,  0.4644, -0.3228,  0.3940, -0.3953,  0.3951, -0.5496,
         0.2693, -0.7602, -0.3508,  0.2334, -1.3239, -0.1694,  0.3938, -0.1026,
         0.0460, -0.6995,  0.1549,  0.5628,  0.3011,  0.3425,  0.1073,  0.4651,
         0.1295,  0.0788, -0.0492, -0.5638,  0.1465, -0.3890, -0.0715,  0.0649,
         0.2768,  0.3279,  0.5682, -1.2640, -0.8368, -0.9485,  0.1358,  0.2727,
         0.1841, -0.5325,  0.3507, -0.0827, -1.0248, -0.6912, -0.7711,  0.2612,
         0.4033, -0.4802, -0.3066,  0.5807, -1.3325,  0.4844, -0.8160,  0.2386,
         0.2300,  0.4979,  0.5553,  0.5230, -0.2182,  0.0117, -0.5516,  0.2108],
        requires_grad=True)
```

[6] print("Bias :", first_conv.bias)

- inspect the bias values used in the model convolutional layer.

Process the feature maps

```
[7]: ### START CODE HERE ###
img = cv2.imread('images/duck2.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
### END CODE HERE ###
```



[7] Read image, convert BGR to RGB, and display image

```
[8]: ### START CODE HERE ###
# Define the mean and std for VGG16 normalization
mean=[0.48235, 0.45882, 0.40784]
std=[0.00392156862745098, 0.00392156862745098, 0.00392156862745098]

# Define the transformation for normalization
transform = transforms.Compose([
    transforms.ToTensor(), # Convert image to tensor : Tensor เป็นโครงสร้างข้อมูลที่ใช้ในการเก็บข้อมูลเชิงตัวเลขหลายมิติ ซึ่งเหมาะสำหรับการแทนข้อมูลภาพ
    transforms.Normalize(mean=mean, std=std) # Normalize image
])

# Define the transformation for normalization
def normalize_image(image):
    # Convert image to float and normalize to [0, 1] range
    image = image.astype(np.float32) / 255.0

    # Convert the image from HWC to CHW format -> PyTorch use CHW
    image = np.transpose(image, (2, 0, 1))

    # Normalize the image using the VGG16 parameters
    image[0] = (image[0] - mean[0]) / std[0]
    image[1] = (image[1] - mean[1]) / std[1]
    image[2] = (image[2] - mean[2]) / std[2]

    # Convert to PyTorch tensor
    image_tensor = torch.from_numpy(image).float()

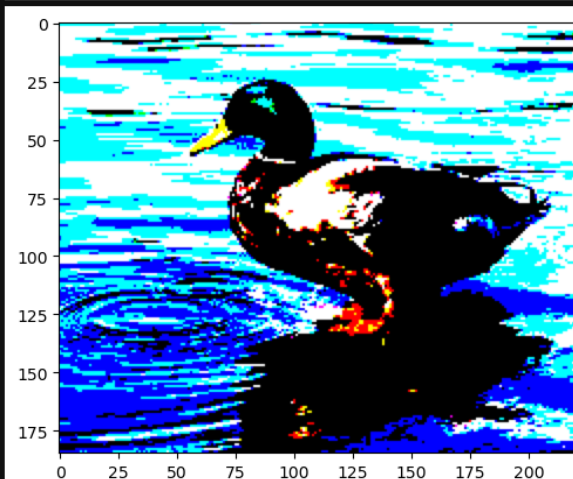
    return image_tensor

# Preprocess the image
image_tensor = normalize_image(img)

# Convert the tensor back to NumPy array for display
# Clip values to [0, 1] range
image_tensor_display = image_tensor.permute(1, 2, 0).numpy() #permute เป็นเมธอดที่ถูกรวบรวมมาโดยเฉพาะสำหรับการสลับมิติของ tensor ใน PyTorch
image_tensor_display = np.clip(image_tensor_display, 0, 1) #ปรับค่าให้แน่ใจว่าอยู่ในช่วง [0, 1]

# Display the image
plt.imshow(image_tensor_display)
plt.show()

### END CODE HERE ###
```



[8] การกำหนดค่า Mean และ Std สำหรับการ Normalization แบบ VGG16:

mean และ std: ค่าคงที่ที่ใช้ในการปรับค่าความเข้มของสีของภาพให้เป็นมาตรฐานตามที่โมเดล VGG16 ถูกฝึกมา ค่าเหล่านี้จะช่วยให้โมเดลสามารถทำนายผลลัพธ์ได้ดีขึ้น โดยให้:

- mean = [0.48235, 0.45882, 0.40784]
- std = [0.00392156862745098, 0.00392156862745098, 0.00392156862745098]
- Combines several image transformations into one pipeline (การกำหนด Transformation สำหรับ Normalization)
 - **transforms.Compose:** ฟังก์ชันที่ใช้ในการรวมขั้นตอนการแปลงภาพหลายๆ ขั้นตอนเข้าด้วยกัน

- **transforms.ToTensor():** แปลงภาพจากรูปแบบ PIL Image หรือ NumPy array เป็น PyTorch Tensor
- **transforms.Normalize(mean=mean, std=std):** ทำการ Normalization ภาพ โดยใช้ค่า mean และ std ที่กำหนดไว้ (a range suitable for the VGG16 model)
- **def normalize_image(image): -> Function to Normalize Image**
 - image = image.astype(np.float32) / 255.0
 - Converts the image to a float format and normalizes the pixel values to a range between 0 and 1.
 - image = np.transpose(image, (2, 0, 1))
 - Rearranges the dimensions of the image from HWC (Height, Width, Channels) to CHW (Channels, Height, Width), which is the required format for PyTorch tensors.
 - image[0] = (image[0] - mean[0]) / std[0]
 - image[1] = (image[1] - mean[1]) / std[1]
 - image[2] = (image[2] - mean[2]) / std[2]
 - Normalizes each channel of the image. This step adjusts the image data so that it fits the distribution expected by the pre-trained VGG16 model.
 - image_tensor = torch.from_numpy(image).float()
 - Converts the normalized image (in NumPy array format) to a PyTorch tensor, making it ready for use in a PyTorch model.
- **image_tensor = normalize_image(img)**
 - Applies the normalize_image function to prepare image
- image_tensor_display = image_tensor.permute(1, 2, 0).numpy()
 - Reorders the dimensions of the tensor back to HWC format from CHW so that it can be displayed as an image.
- image_tensor_display = np.clip(image_tensor_display, 0, 1)
 - Ensures all values in the image tensor are within the [0, 1] range, which is necessary for displaying the image correctly.
- plt.imshow(image_tensor_display)
- plt.show()
 - Display image

```
▶ ~  
### START CODE HERE ###  
  
# print("image_tensor :",image_tensor.shape)  
  
# Add batch dimension  
# image_tensor = image_tensor.unsqueeze(0) # Shape: [1, 3, height, width] -> มิติที่ 0 แทนจำนวนภาพใน batch (ในกรณีนี้มี 1 ภาพ)  
  
# Ensure tensor data type is float32  
image_tensor = image_tensor.to(torch.float32)  
  
# Print tensor shape and type for verification  
print(f"Tensor shape: {image_tensor.shape}")  
print(f"Tensor dtype: {image_tensor.dtype}")  
  
### END CODE HERE ###  
[9]  
... Tensor shape: torch.Size([3, 185, 221])  
Tensor dtype: torch.float32
```

[9] **image_tensor.to(torch.float32):** แปลงชนิดข้อมูลของ image_tensor ให้เป็น float32

แสดงผลข้อมูล image_tensor

- {image_tensor.shape} จะแสดงขนาด (shape) ของ tensor โดยขนาดจะแสดงเป็นจำนวนของข้อมูลในแต่ละมิติ (ช่องสี, ความสูง, ความกว้าง)

- {image_tensor.dtype} จะแสดงชนิดข้อมูล (data type) ของ tensor

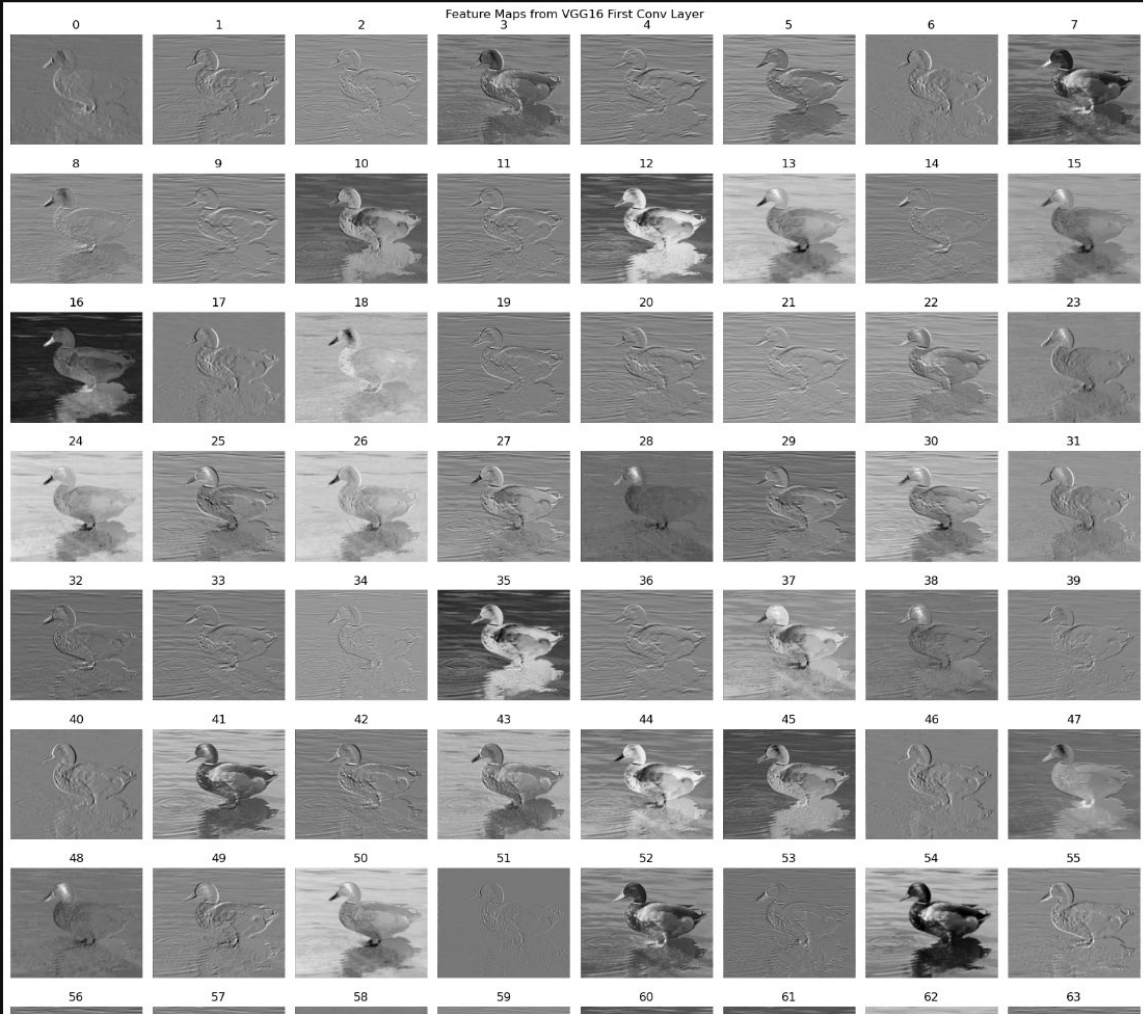
```
[11]: ### START CODE HERE ###  
def plot_featuremap(img, title):  
    """  
    Plot the feature maps from a tensor. Each feature map will be displayed in a subplot.  
  
    Args:  
        img (torch.Tensor): The tensor containing feature maps with shape [batch_size, num_channels, height, width].  
        title (str): The title of the plot.  
    """  
    # Get the number of feature maps  
    num_channels = img.size(0)  
  
    # Define the grid size  
    grid_size = int(np.ceil(np.sqrt(num_channels)))  
  
    # Create a new figure  
    fig, axes = plt.subplots(grid_size, grid_size, figsize=(grid_size * 2, grid_size * 2))  
  
    # Flatten the axes array for easy iteration  
    axes = axes.flatten()  
  
    # Plot each feature map  
    for i in range(num_channels):  
        feature_map = img[i].detach().cpu().numpy()  
  
        axes[i].imshow(feature_map, cmap='gray')  
        axes[i].set_title(f'{i}')  
        axes[i].axis('off') # Hide axes  
  
    # Hide any unused subplots  
    for j in range(num_channels, len(axes)):  
        axes[j].axis('off')  
  
    # Set the title for the entire plot  
    plt.suptitle(title)  
    plt.tight_layout()  
    plt.show()  
    ### END CODE HERE ###  
  
Pass the image to the first convolutional layer and display the feature map output using your plot_featuremap() function.
```

[10] **def plot_featuremap(img, title):** -> function to visualize the feature maps (outputs of convolutional layers) in a grid format.

- num_channels = img.size(0)
 - gives the number of channels โดยใช้ .size(0) เพื่อดึงขนาดของมิติที่ 0 ของ tensor ซึ่งในกรณีนี้คือจำนวน Feature Maps
- grid_size = int(np.ceil(np.sqrt(num_channels)))
 - Calculates the size of the grid needed to display all feature maps. The grid is arranged in a square format, so the number of rows and columns is determined by taking the square root of the number of channels and rounding up to the nearest integer using np.ceil().
- fig, axes = plt.subplots(grid_size, grid_size, figsize=(grid_size * 2, grid_size * 2))
 - created new figure
- axes = axes.flatten()
 - Converts the 2D array into a 1D array to make it easier to iterate (แปลงข้อมูลของ Axes จากโครงสร้างตารางให้กลายเป็น Array เพื่อสะดวกในการวน Loop)
- for i in range(num_channels):
 - feature_map = img[i].detach().cpu().numpy()
 - ดึงข้อมูลของ Feature Map ที่ตำแหน่ง i จาก tensor โดยใช้ .detach() เพื่อแยก Feature Map ออกจาก Graph ของการคำนวณ , .cpu() เพื่อย้ายข้อมูลไปยัง CPU และ .numpy() เพื่อแปลงข้อมูลเป็นแบบ NumPy Array
 - axes[i].imshow(feature_map, cmap='gray')
 - แสดงผล Feature Map บน Axes ที่ตำแหน่ง i โดยใช้สีแบบ grayscale
 - axes[i].set_title(f'{i}')
 - ตั้งค่า Title สำหรับ Axes ที่ตำแหน่ง i โดยแสดงหมายเลขของ Feature Map
 - axes[i].axis('off')
 - ปิดการแสดง Axes ของแต่ละ Feature Map
- for j in range(num_channels, len(axes)): วน Loop เพื่อปิดการใช้งาน Subplots ที่เหลือเกินกว่าจำนวน
 - axes[j].axis('off')
 - ปิดการแสดง Axes ของ Subplots ที่ไม่ต้องการใช้
- plt.suptitle(title): ตั้งค่า Title สำหรับกราฟทั้งหมด โดยใช้ข้อความที่ส่งผ่าน Argument title
- plt.tight_layout(): ปรับ Layout ของกราฟให้เหมาะสม
- plt.show(): แสดงผลกราฟ


```
[22]: ### START CODE HERE ###
#Forward pass through the model to get feature maps
with torch.no_grad():
    # Extract feature maps from the first convolutional layer
    feature_maps = vgg16.features[0](image_tensor) # Assuming we want to visualize the output of the first conv layer

# Plot the feature maps
plot_featuremap(feature_maps, 'Feature Maps from VGG16 - First Conv Layer')
### END CODE HERE ###
```



[22] **with torch.no_grad():**

- disable gradient calculation during the forward pass. This is done because we're only interested in the output (feature maps) and don't need to compute gradients, which saves memory and speeds up computation.

feature_maps = vgg16.features[0](image_tensor)

- Extract feature maps from the first convolutional layer

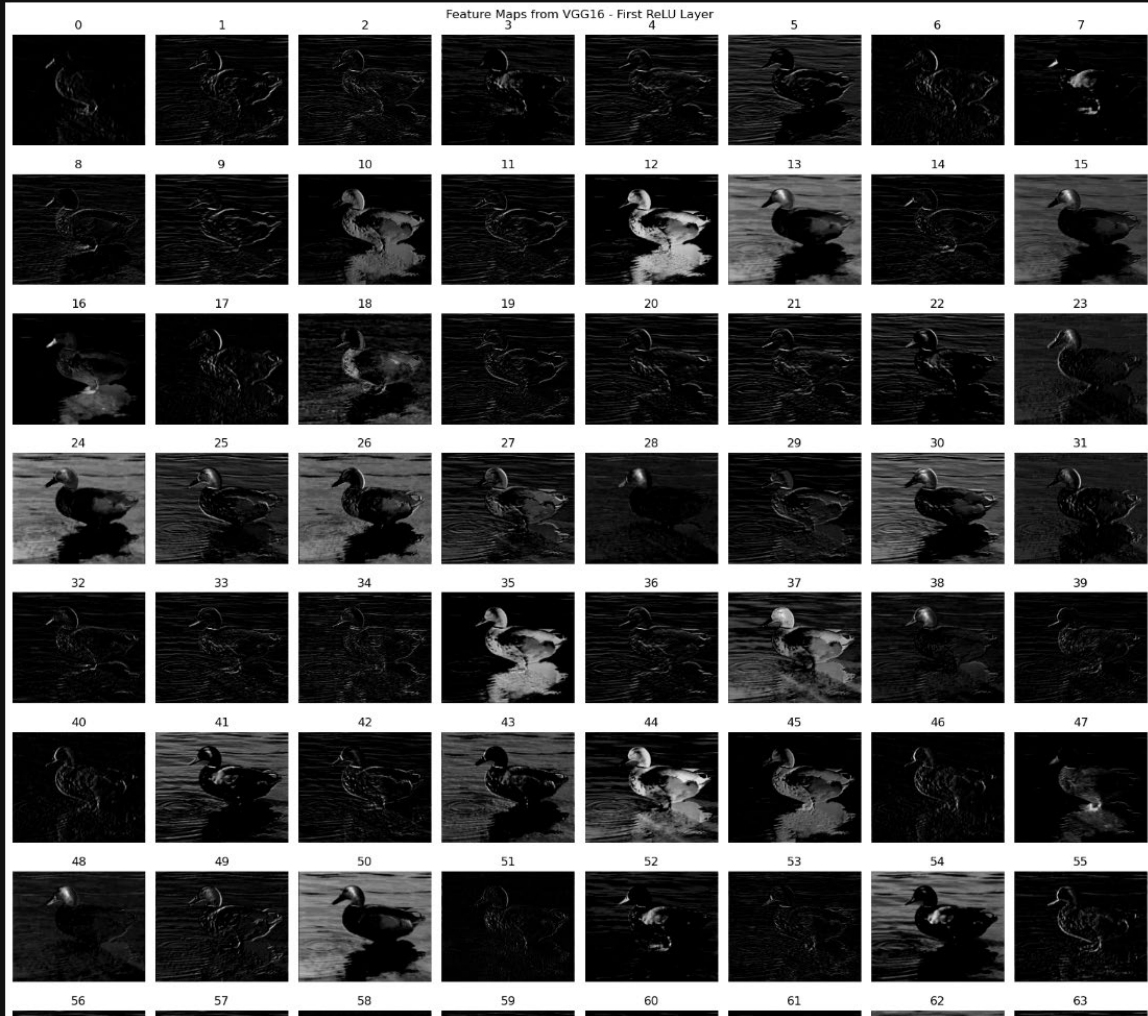
plot_featuremap(feature_maps, 'Feature Maps from VGG16 - First Conv Layer')

- Calls the plot_featuremap function to visualize the feature maps generated by the first convolutional layer of the VGG16 model.

[28]: `### START CODE HERE ###`

```
with torch.no_grad():
    relu_output = vgg16.features[1](feature_maps)

# Plot the feature maps
plot_featuremap(relu_output, 'Feature Maps from VGG16 - First ReLU Layer')
### END CODE HERE ###
```



[28] `relu_output = vgg16.features[1](feature_maps)`

- Extract the second layer of the VGG16 model, which is the ReLU (Rectified Linear Unit) activation layer.

`plot_featuremap(relu_output, 'Feature Maps from VGG16 - First ReLU Layer')`

- Calls the `plot_featuremap` function to visualize the effect of the ReLU activation

Convolution 2D from scratch : การเขียนคอนโวลูชัน 2D ด้วยตัวเอง

```
[29]: ### START CODE HERE ###
def convolution2d(img, kernel, padding, stride):
    """
    Perform a 2D convolution operation on an image with a given kernel.

    Args:
        img (torch.Tensor): Input image tensor of shape [C, H, W].
        kernel (torch.Tensor): Kernel tensor of shape [out_channels, C, kernel_height, kernel_width].
        padding (int): Padding to apply to the input image.
        stride (int): Stride of the convolution operation.

    Returns:
        torch.Tensor: Output feature map tensor.
    """
    # Get dimensions
    # _, C, H, W = img.shape
    C, H, W = img.shape
    out_channels, C_k, kernel_height, kernel_width = kernel.shape

    # Compute output dimensions
    out_height = (H + 2 * padding - kernel_height) // stride + 1
    out_width = (W + 2 * padding - kernel_width) // stride + 1

    # Add padding to the image manually
    img_padded = torch.zeros((C, H + 2 * padding, W + 2 * padding))
    img_padded[:, padding:padding + H, padding:padding + W] = img

    # Initialize output tensor
    output = torch.zeros((out_channels, out_height, out_width))

    print(out_channels, out_height, out_width)

    # Perform convolution
    for oc in range(out_channels):
        for i in range(out_height):
            for j in range(out_width):
                # Define the start and end of the convolution window
                start_i = i * stride
                start_j = j * stride
                end_i = start_i + kernel_height
                end_j = start_j + kernel_width
                |
                # Extract the patch from the padded image
                img_patch = img_padded[:, start_i:end_i, start_j:end_j]

                # Compute the convolution operation for each output pixel
                output[oc, i, j] = torch.sum(img_patch * kernel[oc])

            if (int(oc%10)==0):
                print(oc)

    return output
### END CODE HERE ###
```

[29] **def convolution2d(img, kernel, padding, stride):**

- function performs a 2D convolution operation on an input image tensor using a specified kernel, padding, and stride.

Arguments:

- **img:** Tensor ที่แสดงถึงรูปภาพขนาด [C, H, W] โดย C คือจำนวนช่องสี, H คือความสูง, W คือความกว้าง
- **kernel:** Tensor ที่แสดงถึง Kernel ขนาด [out_channels, Channel, kernel_height, kernel_width] โดย out_channels คือจำนวน Feature Maps ที่จะสร้าง, C_k คือจำนวนช่องสีของ Kernel, kernel_height, kernel_width คือขนาดของ Kernel
- **padding:** จำนวน Pixel ที่จะเพิ่มรอบขอบของรูปภาพ
- **stride:** ระยะห่างที่ Kernel จะเคลื่อนที่ในแต่ละขั้นตอน

- $C, H, W = \text{img.shape}$: รับขนาดของรูปภาพ
- $\text{out_channels}, C_k, \text{kernel_height}, \text{kernel_width} = \text{kernel.shape}$: รับขนาดของ Kernel
- Get Dimensions: กำหนดขนาดของ Feature Map
 - $\text{out_height} = (H + 2 * \text{padding} - \text{kernel_height}) // \text{stride} + 1$
 - $\text{out_width} = (W + 2 * \text{padding} - \text{kernel_width}) // \text{stride} + 1$
 - Compute Output Size -> determine the size of the output feature map after the convolution operation.
- $\text{img_padded} = \text{torch.zeros}((C, H + 2 * \text{padding}, W + 2 * \text{padding}))$: สร้าง Tensor ขนาดใหม่สำหรับรูปภาพที่มี Padding
- $\text{img_padded}[:, \text{padding}:\text{padding} + H, \text{padding}:\text{padding} + W] = \text{img}$: เพิ่ม Padding รอบขอบของรูปภาพ
 - Add padding to the image
 - creates a zero-filled tensor with the padded dimensions.
 - place the original image in the center of this padded tensor, leaving the padding filled with zeros
- $\text{output} = \text{torch.zeros}((\text{out_channels}, \text{out_height}, \text{out_width}))$
 - initialized a zero-filled tensor to hold the results of the convolution operation. dimensions matching the expected output feature map size.

Perform convolution

- วงวน Loop เพื่อทำ Convolution
 - The loops iterate over each output channel and over the spatial dimensions of the output feature map.
 - for oc in $\text{range}(\text{out_channels})$: วงวน Loop สำหรับแต่ละ Feature Map
 - for i in $\text{range}(\text{out_height})$: วงวน Loop สำหรับแต่ละ Pixel ในแนวตั้งของ Feature Map
 - for j in $\text{range}(\text{out_width})$: วงวน Loop สำหรับแต่ละ Pixel ในแนวนอนของ Feature Map
- Convolution Window
 - $\text{start_i} = i * \text{stride}$
 - $\text{start_j} = j * \text{stride}$
 - $\text{end_i} = \text{start_i} + \text{kernel_height}$
 - $\text{end_j} = \text{start_j} + \text{kernel_width}$
 - 'start_i' and 'start_j' calculate the starting indices of the current window on the input image, determined by the stride.
 - 'end_i' and 'end_j' determine the end indices of this window.
- Extracting Patches

- `img_patch = img_padded[:, start_i:end_i, start_j:end_j]`
 - extracts a patch from the padded image corresponding to the current position of the convolution window.
- Convolution Calculation
 - `output[oc, i, j] = torch.sum(img_patch * kernel[oc])`
 - computes the convolution for the current window
- return output คืนค่า Tensor ที่แสดงถึง Feature Map

```
[30]: ### START CODE HERE ###
# image_tensor = image_tensor.unsqueeze(0).squeeze(0) # Remove the batch dimension for our custom convolution function

weights = first_conv.weight.data # Shape: [out_channels, in_channels, kernel_height, kernel_width]
biases = first_conv.bias.data

print(image_tensor.shape)

# Perform convolution using the implemented function
padding = first_conv.padding[0] # Typically, padding is set to 1 for VGG16
stride = first_conv.stride[0] # Typically, stride is set to 1 for VGG16

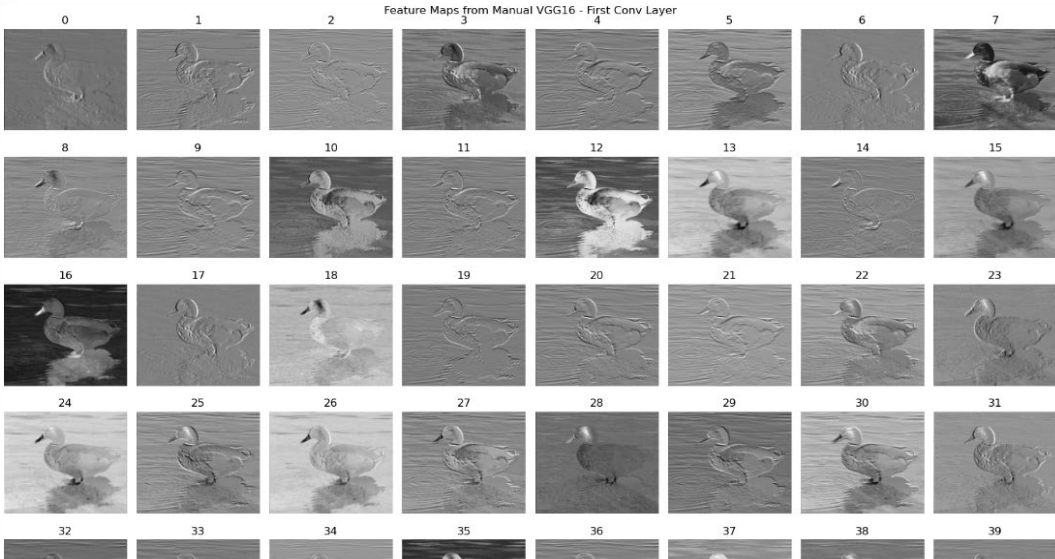
# padding = 0 # Reduced padding
# stride = 2 # Increased stride

feature_maps = convolution2d(image_tensor, weights, padding, stride)

plot_featuremap(feature_maps, 'Feature Maps from Manual VGG16 - First Conv Layer')

### END CODE HERE ###

torch.Size([3, 185, 221])
64 185 221
0
10
20
30
40
50
60
```



[30] Extracts the weight & bias values

- `weights = first_conv.weight.data`: ดึงข้อมูลของน้ำหนัก (weights) จากเลเยอร์ Convolutional แรกของโมเดล VGG16 โดยเก็บไว้ในตัวแปร `weights`
- `biases = first_conv.bias.data`: บรรทัดนี้ดึงข้อมูลของ bias จากเลเยอร์ Convolutional แรกของโมเดล VGG16 โดยเก็บไว้ในตัวแปร `biases`

- `print(image_tensor.shape)`: บรรทัดนี้แสดงขนาดของ `image_tensor` หลังจากปรับแต่งมิติ

Set Padding and Stride Values

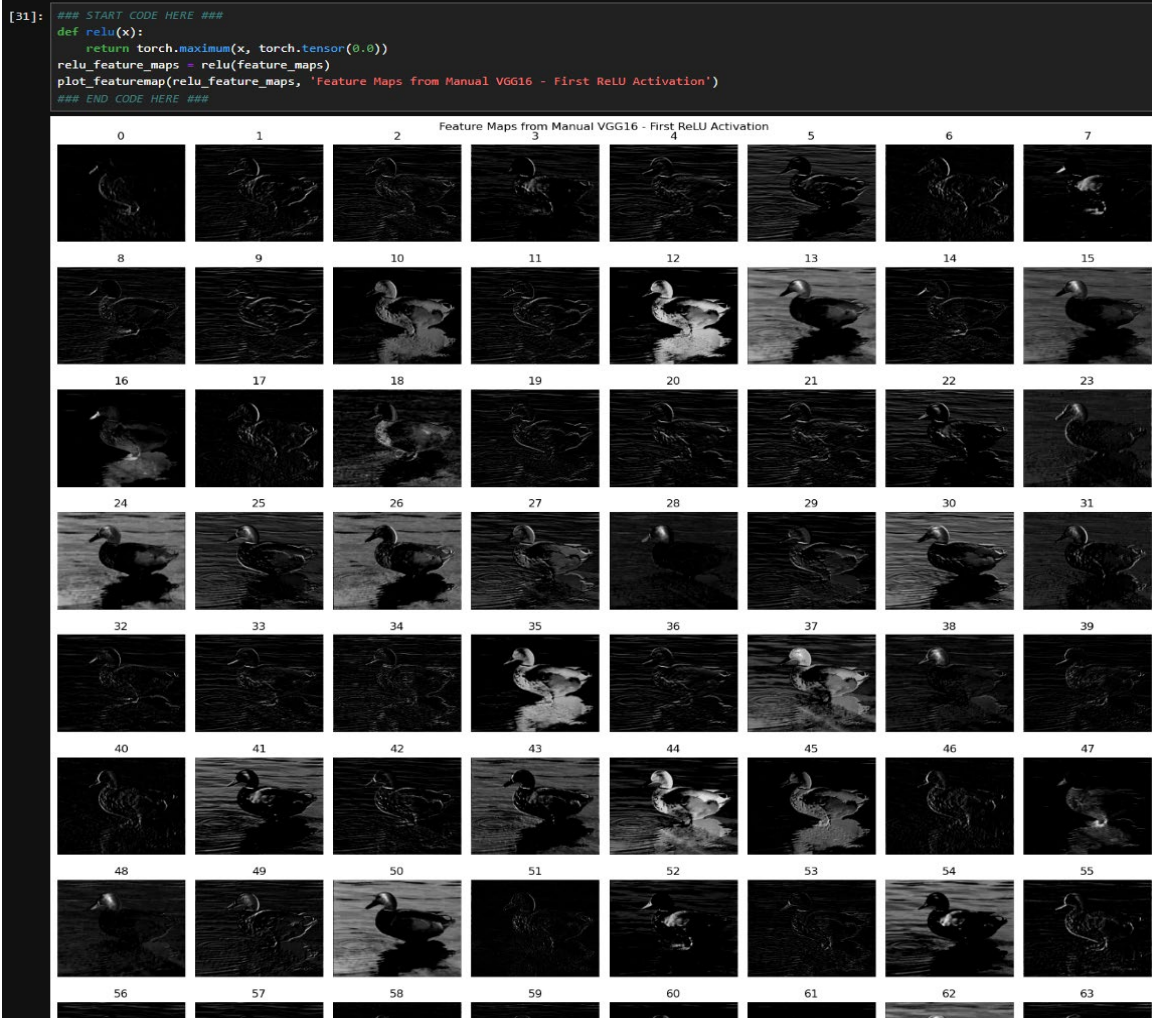
- `padding = first_conv.padding[0]`: บรรทัดนี้ดึงค่าของ Padding ที่ใช้ในเลเยอร์ Convolutional แรกของโมเดล VGG16
- `stride = first_conv.stride[0]`: บรรทัดนี้ดึงค่าของ Stride ที่ใช้ในเลเยอร์ Convolutional แรกของโมเดล VGG16

Perform Convolution Using the convolution2d Function

- `feature_maps = convolution2d(image_tensor, weights, padding, stride)`
 - บรรทัดนี้เรียกใช้ฟังก์ชัน `convolution2d` ที่เขียนเอง โดยส่ง `image_tensor`, `weights`, `padding`, และ `stride` เป็น Argument
 - ผลลัพธ์ที่ได้จะถูกเก็บไว้ในตัวแปร `feature_maps` ซึ่งเป็น Tensor ที่แสดงถึง Feature Maps

Plot the Feature Maps

- `plot_featuremap(feature_maps, 'Feature Maps from Manual VGG16 - First Conv Layer')`



[31] **ReLU Function:**

- def relu(x): ฟังก์ชัน relu รับ Tensor x เป็นอินพุตและคืนค่า Tensor ที่ผ่านการ ReLU Activation
- return torch.maximum(x, torch.tensor(0.0))
 - ภายในฟังก์ชัน torch.maximum จะเปรียบเทียบค่าใน Tensor x กับ 0.0 และคืนค่าที่มากกว่า นั่นหมายความว่าค่าที่เป็นลบจะถูกเปลี่ยนเป็น 0 ส่วนค่าที่เป็นบวกจะคงเดิม 0

Apply the custom ReLU Function:

- relu_feature_maps = relu(feature_maps): เรียกใช้ function: relu เพื่อประยุกต์ใช้ ReLU Activation บน Feature Maps ที่ได้จากการคำนวณ Convolution 2D โดยผลลัพธ์จะถูกเก็บไว้ในตัวแปร relu_feature_maps

Plot the Feature Maps After ReLU Activation:

- plot_featuremap(relu_feature_maps, 'Feature Maps from Manual VGG16 - First ReLU Activation'):
เรียกใช้ function: plot_featuremap เพื่อ plot: Feature Maps ที่ผ่านการ ReLU Activation

โดยที่ภาพที่ผ่านการประมวลผลด้วย ReLU, จะเห็นว่าพื้นที่ที่มีค่า pixel ลบจะหายไป และเฉพาะพื้นที่ที่มีค่า pixel บวกเท่านั้นที่จะถูกเก็บไว้ในภาพ

สรุป:

- ผลลัพธ์ที่ได้จากการทำ Convolution และ Relu เอง(เขียนโค้ดเพื่อดำเนินการ Convolution และ Relu บนภาพโดยตรง) เมื่อเทียบกับผลลัพธ์ที่ได้จากการนำภาพป้อนเข้า model VGG16 (VGG16 ซึ่งเป็น โมเดล Convolutional Neural Network) จะพบว่า Feature Maps ที่ออกจาก layer ที่ 1 (Convolution) และ 2 (Relu) ของทั้ง 2 วิธีให้ผลลัพธ์ที่ตรงกัน ทำให้เราสามารถปรับแต่งให้เหมาะสมกับงานเฉพาะทางได้