Seminar 5 - Lectures 5b and 6a
Computer Vision 1, Master AI, 2025

Arnoud Visser, Martin Oswald, Qi Bi, Roan van Blanken

1 Exercise 1: Convolutional Layer Arithmetics

In this exercise, you will learn the relationships between fully connected (FC) and convolution layers. It is important to understand the advantages and downsides of these two fundamental layers in neural networks.

Q.1.a Given an image I with dimensions $H \times W \times C$ as $100 \times 100 \times 1$, a fully connected (FC) layer, $\mathcal{F} : (H \times W \times C) \rightarrow D$, is used to map the *whole* image (I) into a feature vector $v \in \mathbb{R}^D$ with $D = 1000$. How many parameters are in the FC layer (bias is included)?

Answer: The image I has dimensions $100 \times 100 \times 1$, which means the total number of features (or inputs) to the fully connected (FC) layer is:

$$100 \times 100 \times 1 = 10,000$$

The FC layer maps these 10,000 input features to an output feature vector of dimension $D = 1000$. To calculate the total number of parameters in the FC layer (including the bias), we use the formula:

Number of parameters = (number of input features $\times$ number of output features)+number of biases

Here, the number of input features is 10,000, the number of output features is 1,000, and each output feature has a bias term. Thus, the total number of parameters is:

$$10,000 \times 1,000 + 1,000 = 10,000,000 + 1,000 = 10,001,000$$

Q.1.b Since using $I \in \mathbb{R}^{100\times100\times1}$ as the input of a FC layer is computationally expensive, we can divide the image I into patches of 10×10 pixels, and then apply the same FC layer to each patch. The FC layer outputs a feature vector $v \in \mathbb{R}^D$ for each of its inputs, where $D = 1000$. Calculate the number of parameters in a FC layer. Assuming that there are no overlapping patches, how many feature vectors are produced from the FC layer?

Answer: The image $I \in \mathbb{R}^{100\times100\times1}$ is divided into non-overlapping patches of 10×10 pixels. This results in:

$$\frac{100 \times 100}{10 \times 10} = 10 \times 10 = 100 \text{ patches}$$

Each patch has dimensions $10 \times 10 \times 1$, so the total number of features (or inputs) for each patch is:

$$10 \times 10 \times 1 = 100$$

The FC layer maps these 100 input features to an output feature vector of dimension $D = 1000$. To calculate the total number of parameters in the FC layer (including the bias), we use the formula:

Number of parameters = (number of input features$\times$number of output features)+number of biases

Here, the number of input features is 100, the number of output features is 1,000, and each output feature has a bias term. Thus, the total number of parameters is:

$$100 \times 1,000 + 1,000 = 100,000 + 1,000 = 101,000.$$

Q.1.c Describe the setting of a convolutional layer (kernel size, stride, padding) which produces output with the same dimension as the FC layer described in **Q.1.a** when applied to image I.

Answer: The FC layer treats the *whole* image as its input features. To achieve the same output using a convolutional layer, the kernel size must cover the entire image. Given that the image has dimensions 100×100, the kernel size for the convolutional layer should be:

$$100 \times 100.$$

Since the FC layer takes an input with 1 channel and produces an output feature vector of dimension $D = 1000$, the convolutional layer must have 1 input channel and 1000 output channels. To ensure the output is a single feature vector (similar to the FC layer), the stride should be set to:

$$\text{stride} = 1.$$

No padding is needed since the kernel size matches the input image size. Therefore, the settings for the convolutional layer are:

$$\text{Input channels} = 1, \quad \text{Kernel size} = 100 \times 100, \quad \text{Stride} = 1$$

$$\text{Output channels} = 1000, \quad \text{Padding} = 0$$

Q.1.d Describe the setting of a convolution layer (kernel size, stride, padding) which produces output with the same dimension as the FC layer described in **Q.1.b** when applied to the image I.

Answer: The FC layer treats each of the non-overlapping patches (10×10 pixels) as its input features. To achieve the same output using a convolutional layer, the kernel size must match the patch size. Given that each patch has dimensions 10×10, the kernel size for the convolutional layer should be:

$$10 \times 10.$$

Since the FC layer produces an output feature vector of dimension $D = 1000$ for each patch, the convolutional layer must have 1 input channel (as the image has 1 channel) and 1000 output channels. To ensure the convolutional layer processes non-overlapping patches, the stride should be set to:

$$\text{stride} = 10.$$

No padding is needed because the patches do not overlap, and the kernel size exactly matches the patch size. Therefore, the settings for the convolutional layer are:

$$\text{Input channels} = 1, \quad \text{Kernel size} = 10 \times 10, \quad \text{Stride} = 10$$

$$\text{Output channels} = 1000, \quad \text{Padding} = 0.$$

Now that you have an understanding of the differences and similarities between the fully connected (FC) layer and the convolutional (CONV) layer, you might want to explore how these two types of layers can be connected.[1]

Next, let's move on to the setting of a convolutional layer, as shown in Figure 1.

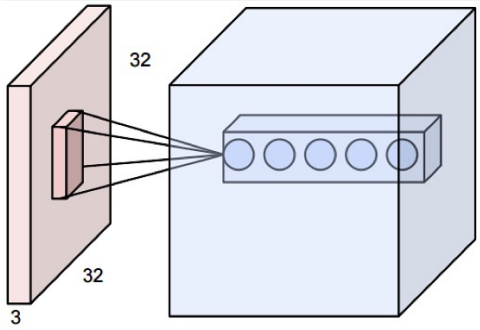

Figure 1: Input size: $32 \times 32 \times 3$, receptive field: 5×5, number of output features: 5 along the depth dimension

Q.1.e What is the size of the output volume with stride 1? (Without padding)

Answer: Given an input size of $32 \times 32 \times 3$, a convolutional layer with a receptive field (kernel size) of 5×5, and a stride of 1, we want to find the size of the output volume without using padding. To calculate the output size along one spatial dimension (height or width), we use the formula:

$$\text{Output size} = \frac{\text{Input size} - \text{Kernel size}}{\text{Stride}} + 1$$

Plugging in the values:

$$\text{Output size} = \frac{32 - 5}{1} + 1 = 28$$

Since the convolutional layer has 5 output features along the depth dimension, the final output volume will have the dimensions:

$$28 \times 28 \times 5$$

Q.1.f How many weights are learned in one filter? How many in this layer?

Answer: For one filter (kernel) in this convolutional layer, the number of weights is calculated by multiplying the kernel size with the number of input channels:

$$5 \times 5 \times 3 = 75$$

Since this layer has 5 filters, the total number of weights in this layer is:

$$75 \times 5 = 375$$

Note: Often, discussions about convolution ignore the bias term. However, this is not entirely accurate, as frameworks like PyTorch include a learnable bias by default. If we include the bias, the number of weights for one filter becomes:

$$5 \times 5 \times 3 + 1 = 76$$

Thus, the total number of parameters in this layer, including the bias, is:

$$5 \times 76 = 380$$

[1] For more details on the connection between FC and CONV layers, refer to the **Converting FC layers to CONV layers** section in this comprehensive note.

Q.1.g What is the depth of a filter in the next layer?

Answer: The depth of a filter in the next layer is equal to the number of output channels from the previous layer. Since the current layer has 5 output channels, the depth of a filter in the next layer will be:

$$5$$

Q.1.h What is the size of the output volume with stride 3?

Answer: Given an input size of $32 \times 32 \times 3$, a receptive field (kernel size) of 5×5, and a stride of 3, the output size can be calculated using the formula:

$$\text{Output size} = \frac{\text{Input size} - \text{Kernel size}}{\text{Stride}} + 1.$$

Plugging in the values:

$$\text{Output size} = \frac{32 - 5}{3} + 1 = 10.$$

Since there are 5 output channels, the size of the output volume is:

$$10 \times 10 \times 5.$$

Q.1.i How many weights are learned with stride 3?

Answer: The number of weights learned in the convolutional layer does not change with the stride because the weights are associated with the kernel size and the input channels, not the number of positions where the kernel is applied. Each filter (or kernel) has a fixed size, in this case, $5 \times 5 \times 3$, and learns a specific set of parameters (including the bias).

Changing the stride only affects the number of times the kernel is applied across the input, which influences the size of the output volume. However, it does not change the total number of weights within each filter. As discussed in Q.1.f, the total number of weights (including biases) is:

$$5 \times 76 = 380.$$

2 Exercise 2: Weight Sharing

Q.2.a What are the properties and/or advantages of weight sharing in a convolutional layer?

Answer: Weight sharing is a crucial factor for the success of convolutional layers in visual recognition tasks. It significantly reduces the number of parameters in neural networks, which allows for building very deep architectures, such as ResNet or EfficientNet. Additionally, weight sharing introduces an inductive bias of translation equivariance. This means that the network can recognize features regardless of their location in the input image, which is known to be a key factor in visual recognition.

Q.2.b Describe a situation where weight sharing is not beneficial for recognition.

Answer: Weight sharing in convolutional layers assumes that the same features (e.g., edges, textures) are useful across different parts of the input image, based on the property of translation equivariance. However, weight sharing is not beneficial when translation equivariance does not hold. This often occurs in highly structured data, such as face or fine-grained recognition. For example, in face recognition, different filters are needed to process specific parts of a human face (e.g., mouth, eyes, nose) because these features have distinct characteristics and do not appear uniformly across the image.

3 Exercise 3: Gradient

The gradient is an important factor for the successful training of neural networks (read this article if you are not convinced). This part will walk you through some common problems in training convolutional neural networks.

Q.3.a Given an input matrix and a convolutional layer with a weight matrix (kernel):

$$x = \begin{bmatrix} 0 & -1 & 3 \\ 2 & 0.5 & 1 \end{bmatrix} \in \mathbb{R}^{2\times 3\times 1}, \quad W = \begin{bmatrix} 0 & 1 \\ 1 & 0.5 \end{bmatrix} \in \mathbb{R}^{2\times 2\times 1}$$

and no bias. Assume stride $= 1$ and padding $= 0$. Compute the output matrix $o \in \mathbb{R}^{1\times 2\times 1}$.

Answer: The kernel slides over the input with stride 1 and no padding. For the first position (top-left corner of the input):

$$(0 \times 0) + (-1 \times 1) + (2 \times 1) + (0.5 \times 0.5) = 1.25.$$

For the second position (top-right corner of the input):

$$(-1 \times 0) + (3 \times 1) + (0.5 \times 2) + (1 \times 0.5) = 4.$$

Therefore, the output matrix is $\begin{bmatrix} 1.25 & 4 \end{bmatrix} \in \mathbb{R}^{1\times 2\times 1}$.

Q.3.b What is the gradient with respect to the input matrix x, given that the gradient with respect to the output matrix o is

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

The loss L is scalar, and the layer output $\mathbf{o} = [o_1, o_2]^\top$ depends on the input matrix $\mathbf{X} \in \mathbb{R}^{2\times 3}$. By the chain rule,

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{o}} \cdot \frac{\partial \mathbf{o}}{\partial \mathbf{X}} = \sum_k \frac{\partial L}{\partial o_k} \cdot \frac{\partial o_k}{\partial \mathbf{X}}.$$

What's given is the upstream gradient $\frac{\partial L}{\partial o}$. For the Jacobian of the scalar o_k w.r.t. all entries of $\mathbf{X}$, we need to write out the entries of o:

$$\mathbf{o} = \begin{bmatrix} o_1 \\ o_2 \end{bmatrix}^T = \begin{bmatrix} x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22} \\ x_{12}w_{11} + x_{13}w_{12} + x_{22}w_{21} + x_{23}w_{22} \end{bmatrix}^T.$$

Then we have the gradient with respect to the input:

$$\frac{\partial L}{\partial \mathbf{o}} = \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \Rightarrow \quad \frac{\partial L}{\partial \mathbf{X}} = 1 \cdot \frac{\partial o_1}{\partial \mathbf{X}} + 1 \cdot \frac{\partial o_2}{\partial \mathbf{X}} = \begin{bmatrix} w_{11} & w_{11} + w_{12} & w_{12} \\ w_{21} & w_{21} + w_{22} & w_{22} \end{bmatrix}.$$

Alternatively there is a convenient way. The gradient with respect to input x is summarized by the full cross-correlation of the gradient w.r.t. o with the original convolution kernel, which is $o \circledast W$.

We have the W here:

$$W = \begin{bmatrix} 0 & 1 \\ 1 & 0.5 \end{bmatrix}.$$

Note for the full cross-correlation, we slide the matrix o over the kernel, allowing partial overlaps, and obtain the gradient matrix which has the same shape as input x. For the top-left corner, the calculation is $(0 \times 1) = 0$. For the top-middle, it is $(0 \times 0.5) + (1 \times 1) = 1$. For the top-right, the calculation is $(1 \times 1) = 1$. For the bottom-left, it is $(1 \times 1) = 1$. For the bottom-middle, the calculation is $(1 \times 1) + (0.5 \times 1) = 1.5$ and for the bottom-right, it is $(0.5 \times 1) = 0.5$. Therefore, the final gradient of the input matrix x is

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 1.5 & 0.5 \end{bmatrix}.$$

Q.3.c Consider the ReLU non-linearity $z = \max(0, x)$, where the input matrix is $x = \begin{bmatrix} 0 & -1 \\ 2 & 0.5 \end{bmatrix} \in \mathbb{R}^{2\times 2}$. First, calculate the matrix z. Then, given that the gradient w.r.t. z is $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, find the gradient w.r.t. x.

Answer:
Let's first denote:

$$x = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \; z = \begin{bmatrix} f(a) & f(b) \\ f(c) & f(d) \end{bmatrix}.$$

ReLU is the local activation here, similarly from the chain rule, we have:

$$\frac{\partial L}{\partial X_{ij}} = \frac{\partial L}{\partial F(\mathbf{X})_{ij}} \cdot \frac{\partial F(\mathbf{X})_{ij}}{\partial X_{ij}},$$

The difference here from question Q.3.b is that ReLU is applied element-wise to its input tensor, that is $F(\mathbf{X})$ is element-wise, thus:

$$\frac{\partial L}{\partial X_{ij}} = \frac{\partial L}{\partial F(\mathbf{X})_{ij}} \cdot \frac{\partial F(\mathbf{X})_{ij}}{\partial X_{ij}} = \frac{\partial L}{\partial F(\mathbf{X})_{ij}} \cdot \frac{\partial f(X_{ij})}{\partial X_{ij}} = \frac{\partial L}{\partial f(X_{ij})} \cdot f'(X_{ij}),$$

Note we write the above equation for the input entry with index ij, if we put all the elements back to matrix form:

$$\frac{\partial L}{\partial x} = \begin{bmatrix} f'(a)g_{11} & f'(b)g_{12} \\ f'(c)g_{21} & f'(d)g_{22} \end{bmatrix} = \underbrace{\begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}}_{\frac{\partial L}{\partial z}} \odot \underbrace{\begin{bmatrix} f'(a) & f'(b) \\ f'(c) & f'(d) \end{bmatrix}}_{f'(x)} \quad \text{(Element-wise product)}.$$

Lastly, we substitute with the values in the problem setting.

$$x = \begin{bmatrix} 0 & -1 \\ 2 & 0.5 \end{bmatrix}, \quad z = \text{ReLU}(x) = \begin{bmatrix} 0 & 0 \\ 2 & 0.5 \end{bmatrix}.$$

Take $f'(x) = 1[x > 0]$ and $f'(0) = 0$:

$$f'(x) = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \quad \frac{\partial L}{\partial z} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Therefore

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \odot f'(x) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \odot \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}.$$

Q.3.d Read this post on the *Dying* ReLU problem here. Describe and explain possible ways to prevent this problem in training convolutional neural networks.

Answer: The *Dying* ReLU problem occurs when all output neurons are suppressed by the ReLU function, resulting in zero gradients for the network parameters. This makes the network unable to update its parameters, getting stuck in the current state. Solutions include:

- Using normalization techniques (e.g., Batch Normalization)
- Proper weight initialization (e.g., Kaiming or Xavier initialization)
- Using alternative activation functions (e.g., Leaky ReLU, ELU, GeLU)
- Using residual networks with skip connections

Q.3.e A common problem in training deep neural networks is gradient vanishing and exploding, as discussed in this article. List some common solutions for this problem and explain them.

Answer: The gradient can either vanish (decrease) or explode (increase) as it back-propagates through each layer. This can lead to the network weights stopping updates (vanishing) or numerical instability (exploding). Solutions include:

- Normalization (e.g., Batch Normalization) to keep neuron activations in a stable range
- Gradient clipping to limit the gradient values and prevent the exploding gradient problem
- Residual connections to improve gradient flow through the network
- Proper weight initialization (e.g., Kaiming or Xavier) to control the gradient magnitude

4 Exercise 4: Single Shot Object Detection

Q.4.a What are the main differences between Single Shot Detectors like YOLO and older two-stage object detection methods like R-CNN?

Answer: The main differences are as follows:

1. Single Shot Detectors (SSDs) like YOLO perform localization (region proposal), feature extraction, and classification in one unified step. In contrast, two-stage detectors like R-CNN first generate region proposals and then perform feature extraction and classification separately.
2. Single Shot Detectors process the whole image in a single pass through the network. On the other hand, two-stage detectors only pass Regions of Interest (ROIs) into the network for further processing.
3. In Single Shot Detectors, the output vector (also called the prediction vector) contains both the bounding box coordinates and the class confidence/probability for each detected object. Two-stage detectors typically handle these steps separately.

Read sections 1 (Introduction) and 4 (Related Work) of the YOLO v4 paper. The purpose of the questions below is to explore the variety of established methods developed in object detection.

Q.4.b Mention two backbone networks that we discussed in class.

Answer: Some of the backbone networks discussed in class include VGG, ResNet, and Darknet53.

Q.4.c We have discussed feature pyramids in class. How do the authors of the YOLO v4 paper categorize these types of architectural features?

Answer: The authors categorize these architectural features as the "Neck" of the object detection network. The Neck is responsible for combining features from different layers, helping the network detect objects at various scales.

Q.4.d In paragraph 2.2 (Bag of Freebies) of the paper, the authors describe a problem with using box coordinates in the output objective function. What is this problem, and how can it be solved?

Answer: The problem is that the loss increases with the scale of the bounding box. This means that larger objects can dominate the loss function, leading to an imbalance in training. To address this issue, the authors suggest using Intersection over Union (IoU) loss, which normalizes the loss across different scales and focuses on the overlap between the predicted and ground truth boxes.

5 Exercise 5: 3D Convolutions

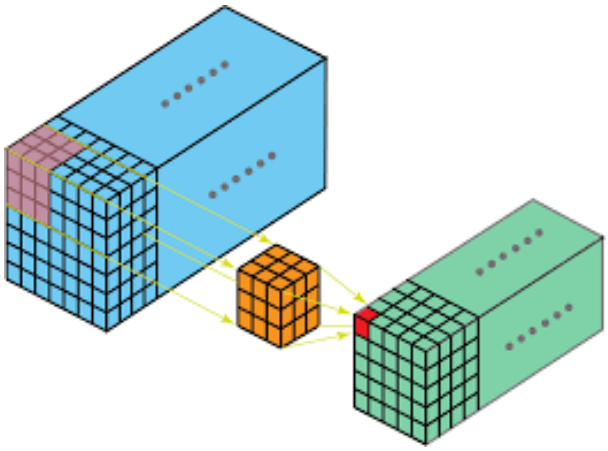

Figure 2: Demonstration of 3D convolution.

Consider a grayscale video with 16 frames, each of size 224×224. You want to generate a feature map with 64 output channels using $3 \times 3 \times 3$ convolutional filters. Proper padding is applied.

Q.5.a What is the dimension of the output weight tensor of the 3D layer? How many parameters are there in total?

Answer: The output weight tensor for the 3D convolutional layer consists of 64 filters (output channels), each with dimensions $3 \times 3 \times 3$ and a single input channel since the input video is grayscale. Therefore, the weight tensor has the shape $[64, 1, 3, 3, 3]$. To calculate the total number of parameters, multiply the number of filters by the size of each filter. This gives $64 \times 1 \times 3 \times 3 \times 3 = 1728$. Additionally, each of the 64 filters has one bias term, resulting in 64 additional parameters. The total number of parameters is $1728 + 64 = 1792$.

Q.5.b What is the dimensionality of the output feature maps if stride = 1 for all 3 dimensions?

Answer: Since the stride is 1 in all dimensions (temporal and spatial) and proper padding is applied, the output feature map will have the same dimensions as the input for the spatial size and number of frames. Therefore, the output will have 64 channels, 16 frames (same as input), and each frame will have dimensions 224×224. The output dimensionality is $[64, 16, 224, 224]$.

Q.5.c What is the dimensionality of the output feature maps if stride = 2 for the temporal dimension and stride = 1 for the spatial dimensions?

Answer: Here, the stride is 2 along the temporal dimension, so the number of frames will be halved. With 16 input frames, the output will have $\frac{16}{2} = 8$ frames. Since the stride is 1 for the spatial dimensions and proper padding is used, the spatial size of each frame remains 224×224. The output feature map will have 64 channels, 8 frames, and each frame will have dimensions 224×224. The output dimensionality is $[64, 8, 224, 224]$.

Q.5.d Suppose that a standard 3D convolution layer takes an input feature matrix F of shape (l_F, w_F, h_F, c_F) and yields a feature matrix G of size (l_G, w_G, h_G, c_G), where c denotes the number of channels. Also suppose the kernel size is $k \times k \times k$. What is the computational cost of a 3D convolution here?

Answer: The computational cost of a 3D convolution is the product of the kernel size, the number of input channels, the number of output channels, and the spatial dimensions of the input feature map. This can be expressed as $k \times k \times k \times c_F \times c_G \times l_F \times w_F \times h_F$. Note that this calculation assumes proper padding is applied, so the input and output spatial dimensions remain the same.

Q.5.e We can compare a 3D convolution with a 2D convolution over each of the video frames. What is an advantage of such (2+1)D convolutions over 3D convolutions?

Answer: (2+1)D convolutions, where separate 2D convolutions are applied over each frame followed by a 1D convolution across the temporal dimension, have fewer parameters compared to 3D convolutions. This results in faster computation, making (2+1)D convolutions a more efficient alternative for certain tasks.