CNN Questions

Introduction to Deep Learning (Technische Universität München)

Problem 4 Convolutional Neural Networks and Receptive Field (12 credits)

A friend of yours asked for a quick review of convolutional neural networks. As he has some background in computer graphics, you start by explaining previous uses of convolutional layers.

a) You are given a two dimensional input (e.g., a grayscale image). Consider the following convolutional kernels

$$C_1 = \frac{1}{9} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad C_2 = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}.$$

0 1 2

What are the effects of the filter kernels C_1 and C_2 when applied to the image?

C_1: Local/box (0.5p) blur/smoothing (0.5p) kernel. Note: If only mean/arg is mentioned instead of blur then 0.5p
C_2: vertical (0.5p) edge detector (0.5p)

After showing him some results of a trained network, he immediately wants to use them and starts building a model in Pytorch. However, he is unsure about the layer sizes so you quickly help him out.

b) Given a Convolution Layer in a network with 5 filters, filter size of 7, a stride of 3, and a padding of 1. For an input feature map of $26 \times 26 \times 26$, what is the output dimensionality after applying the Convolution Layer to the input?

0 1 2

$8 \times 8 \times 5$ (2p) 1p for only kernel size computation $\frac{26-7+2\cdot 1}{3} + 1 = 7 + 1 = 8$

c) You are given a convolutional layer with 4 filters, kernel size 5, stride 1, and no padding that operates on an RGB image.

0 1 2

1. What is the shape of its weight tensor?
2. Name all dimensions of your weight tensor.

Shape: (3, 4, 5, 5) (1p)
Reasoning: input size/rgb channels, output size/channels, width, height (1p)

Note: -1p if only 3 dimensions are mentioned, -1p if 5's are simply described as filter size

Now that he knows how to combine convolutional layers, he wonders how deep his network should be. After some thinking, you illustrate the concept of receptive field to him by these two examples. For the following two questions, consider a grayscale 224x224 image as network input.

d) A convolutional neural network consists of 3 consecutive 3×3 convolutional layers with stride 1 and no padding. How large is the receptive field of a feature in the last layer of this network?

0 1

$1x1 \rightarrow 3x3 \rightarrow 5x5 \rightarrow 7x7$ (1p)
Note: -0.5p if no tuple

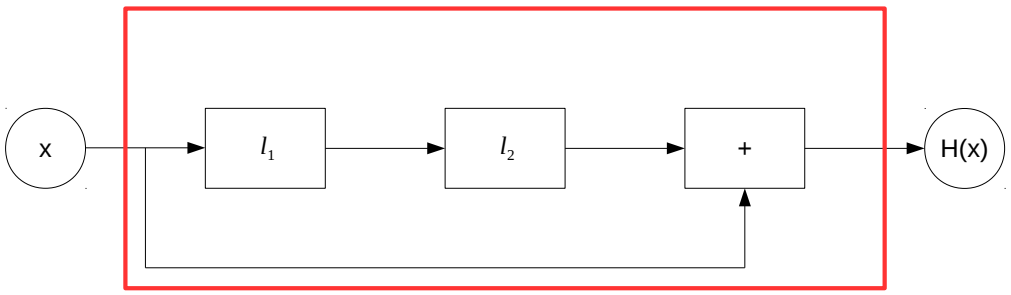
0 1 2

e) Consider a network consisting of a single layer.

1. What layer choice has a receptive field of 1?
2. What layer has a receptive field of the full image input?

1x1 convolution or identity (1p)
fully connected or conv/pooling layer with kernel size equals full input size (224x224) (1p)
If the answer is reasonable but incomplete: -0.5p

Blindly, he stacks 10 convolutional layers together to solve his task. However, the gradients seem to vanish and he can't seem to be able to train the network. You remember from your lecture that ResNet blocks were designed for these purposes.



0 1 2

f) Draw a ResNet block in the image above (1p) containing two linear layers, which you can represent by l_1 and l_2. For simplicity, you don't need to draw any non-linearities. Why does such a block improve the vanishing gradient problem in deep neural networks (1p)?

1p for correct drawing
Note: if image strucutre is correct but: i) arrow is missing or ii) "+" symbol is missing or not drawn correctly 0.5p
1p for highway of gradients
Note: if only forward pass is mentioned then no points

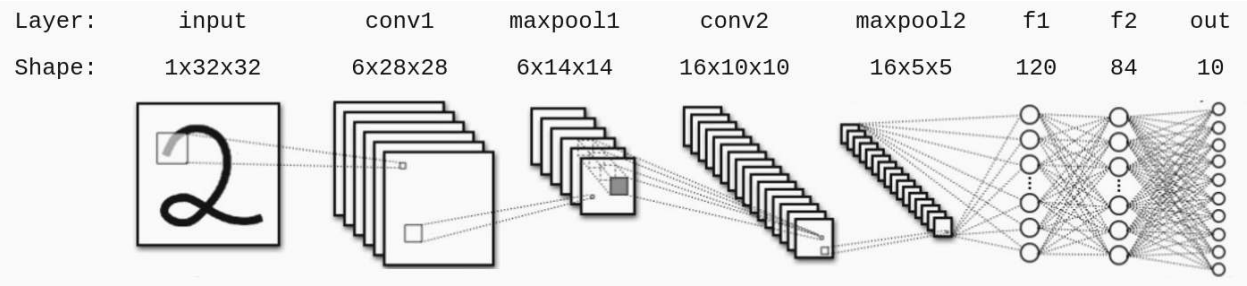
0 1

g) For your above drawing, given the partial derivative of the residual block $R(x) = l_2(l_1(x))$ as $\frac{\partial R(x)}{\partial x} = r$, calculate $\frac{\partial H(x)}{\partial x}$.

$\frac{\partial H(x)}{\partial x} = \frac{\partial x+R(x)}{\partial x} = \frac{\partial x}{\partial x} + \frac{\partial R(x)}{\partial x} = 1 + r$
1p for $\frac{\partial H(x)}{\partial x} = 1 + r$

Problem 7 Convolutional Neural Networks (12 credits)

You are contemplating design choices for a convolutional neural network for the classification of digits. LeCun et. al suggest the following network architecture:



For clarification: the shape **after** having applied the operation 'conv1' (the first convolutional layer in the network) is 6x28x28.
All operations are done with stride 1 and no padding. For the convolution layers, assume a kernel size of 5x5.

0 1 2 3

a) Explain the term 'receptive field' (1p). What is the receptive field of one pixel after the operation 'maxpool1'(1p)? What is the receptive field of a neuron in layer 'f1' (1p)?

Receptive field is the size of the region in the input space that a pixel in the output space is affected by.
maxpool1: 6x6. One pixel after maxpool1 is affected by 4 pixels (2x2) in conv1. with 5x5 kernel and stride 1, a 2x2 output comes from a 6x6 grid.
(0.5p if only maxpool1 wrt to conv1(= 2x2) specified)
f1: whole image (32x32)

0 1

b) Instead of digits, you now want to be able to classify handwritten alphabetic characters (26 characters). What is the **minimal** change in network architecture needed in order to support this?

Change no. of output neurons from 10 to 26
(0.5p if only "change number of output neurons" specified)

0 1 2

c) Instead of taking 32×32 images, you now want to train the network to classify images of size 68×68. List two possible architecture changes to support this?

- Resize layer to downsample images to 32x32 / Downsample images to 32x32 (preprocess)
- conv 5x5 ($68 \rightarrow 64$) + maxpool 2x2 ($64 \rightarrow 32$) before the current architecture (0.5p if only specified conv+maxpool without parameters)
- Change input dimension of layer f1 to 16x14x14 (= 3136) (0.5p if only suggested changing input dimension layer without new dim)
- fully convolutional layers + global average pooling (0.5p if only fully conv layer suggested without global average pooling)

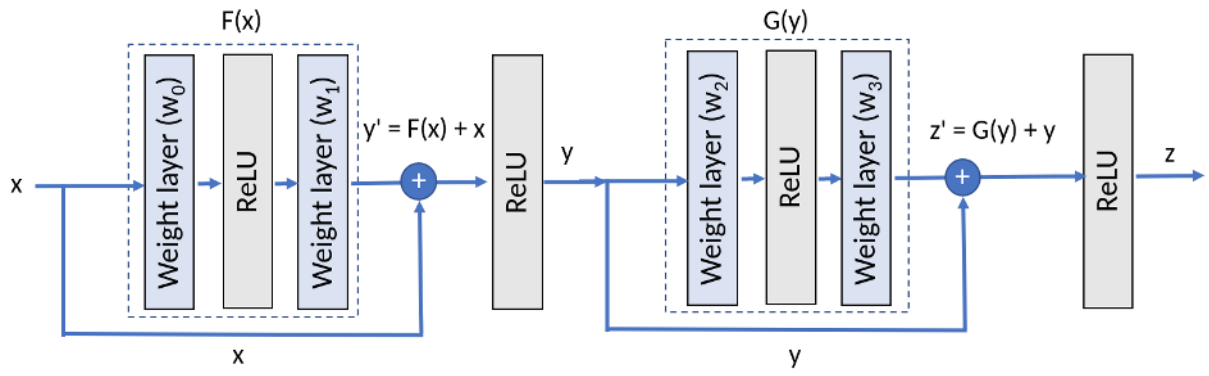
d) Your architecture works and you manage to classify characters fairly well. After reading many online blogs, you decide to try out a much deeper network to boost the network's capacity. Name 2 problems that you might encounter when training very deep networks.

0 1 2

- Vanishing gradients
- Memory issues
- Overfitting
- Increased training time

e) You read that skip connections are beneficial for training deep networks. In the following image you can see a segment of a very deep architecture that uses skip connections. How are skip connections helpful? (1p). Demonstrate this mathematically by computing gradient of output z with respect to 'w_0' for the network below in comparison to the case without a skip connection (3p). For simplicity, you can assume that gradient of ReLU, $\frac{d(ReLU(p))}{dp} = 1$.

0 1 2 3 4



Help prevent vanishing gradients / Provides highway for the gradients in backward pass (1p)

Let,

$$z' = G(y) + y$$
$$G(y) = ReLU(w_2 y) w_3$$
$$z = ReLU(z')$$
$$y' = F(x) + x$$
$$F(x) = w_1 ReLU(w_0 x)$$
$$y = ReLU(y')$$
$$\frac{dz}{dw_0} = \frac{dz}{dz'}\frac{dz'}{dy}\frac{dy}{dy'}\frac{dy'}{dw_0}$$
$$\frac{dz}{dw_0} = \frac{d(ReLU(z'))}{dz'}(\frac{dG(y)}{dy} + 1)\frac{d(ReLU(y'))}{dy'}\frac{dF(x)}{dw_0}$$
$$\frac{dz}{dw_0} = \frac{d(ReLU(z'))}{dz'}(w_3 w_2 \frac{d(ReLU(w_2 y))}{d(w_2 y)} + 1)\frac{d(ReLU(y'))}{dy'} w_1 x \frac{d(ReLU(w_0 x))}{d(w_0 x)})$$

Putting ReLU derivatives to 1

$\frac{dz}{dw_0} = (w_3 w_2 + 1) w_1 x$ (2 points for full expansion, 1pt if $\frac{dG(y)}{dy}$ and $\frac{dF(x)}{dw_0}$ are not expanded)

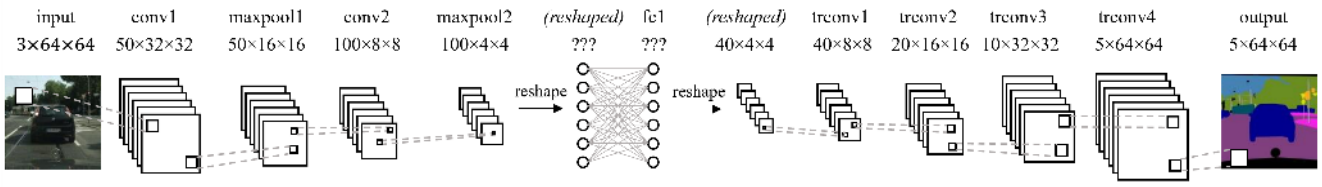
Comparing this to derivative without skip connection, which is

$$\frac{dz}{dw_0} = (w_3 w_2) w_1 x \text{ (1 point / 0.5p if not expanded)}$$

The extra '+1' term in the skip connection derivative help propagation of gradient flow, preventing vanishing gradients

Problem 3 Convolutions (13 credits)

You are asked to perform **per-pixel** semantic segmentation on the Cityscapes dataset, which consists of RGB images of European city streets, and you want to segment the images into 5 classes (vehicle, road, sky, nature, other). You have designed the following network, as seen in the illustration below:
For clarification of notation: The shape **after** having applied the operation 'conv1' (the first convolutional layer in the network) is 50x32x32.
You are using 2D convolutions with: `stride` = 2 , `padding` = 1 , and `kernel_size` = 4 .
For the MaxPool operation, you are using: `stride` = 2 , `padding` = 0 , and `kernel_size` = 2 .



3.1 What is the shape of the weight matrix of the fully-connected layer 'fc1'? (Ignore the bias)

0 1 2

input: $100 \times 4 \times 4 = 1600$
output: $40 \times 4 \times 4 = 640$
weight matrix: 1600×640
(-0.5 if final output is not calculated, -1 if only input OR output is correct and explained, -1 if only input or output size is correct and not explained)

3.2 Explain the term 'receptive field' (1p). What is the receptive field of one pixel of the activation map. after performing the operation 'maxpool1'(1p)? What is the receptive field of a single neuron in the output of layer 'fc1' (1p)?

0 1 2 3

the region in the input space (0.5p) that a pixel in the output space is affected by / that affects a particular unit in the network (0.5). alternative: spatial extent of the connectivity of a convolutional filter (1p)
maxpool1: 6x6. One pixel after maxpool1 is affected by 4 pixels (2x2) in conv1. with 4x4 kernel and stride 2, a 2x2 output comes from a 6x6 grid. (1p)
fc1: whole image (64x64) (1p)

Receptive field is the total area of the image encoded in a single pixel after a convolution operation
Receptive field of one pixel of the activation map: It's the area from which each pixel has gotten its information from after applying the previous convolutions it's the same as the receptive field of its corresponding pixel

0 1

3.3 You now want to be able to classify finer-grained labels, which comprise of 30 classes. What is the **minimal** change in network architecture needed in order to support this without adding any additional layers?

1in - in trconv4/the last layer(0.5p) use 30 channels(0.5p) instead of 5 (trying to add or remove any layer or changing something in the fc layer 0 points as those are not the minimal change)

Use the weights as they are but change and retrain the fully connected layer to classify the 30 classes instead of just 5

0 1 2

3.4 Luckily, you found a pre-trained version of this network, which is trained on the original 5 labels. (It outputs a tensor of shape $5 \times 64 \times 64$). How can you make use of/build upon this pre-trained network (as a black-box) to perform segmentation into 30 classes.

- **add** conv or other layer at the end (with 30 output channels, that preserves size, e.g. 1x1) (1p) **after the pretrained network** to predict the class
Black-box: any attempts to change the pretrained network (e.g to only use part of the network) is incorrect (0p)

Variational auto encoder. By training the network to generate new photos of the same type it will also learn the features of these images

0 1 2 3

3.5 Luckily, you have gained access to a large dataset of city street images. Unfortunately, these images are not labelled, and you do not have the resources to annotate them. However, how can you still make use of these images to improve your network? Explain the architecture of any networks that you will use and explain how training will be performed. (Note: This question is independent of (3.3) and (3.4))

1. Choose the right architecture: autoencoder (or other methods e.g. U-net, VAE)(1p)
[Only answer transfer learning 0.5p, as the question asked about a specific architecture, not type of training method]
2. Training: Train it on this dataset for reconstructing the input image in an unsupervised way (1p).
3. Usage: Then use the trained encoder part to extract features for your network (1p).
-0.5 if feature extraction is not mentioned anywhere in the answer
[GAN is only accepted (1p) If with reasonable explanation for "how"] [Using your network/GAN/KNN to predict the label of the new image is not accepted for this subproblem.]
[Approaches which does not include a network architecture, e.g.K Means: 1p]

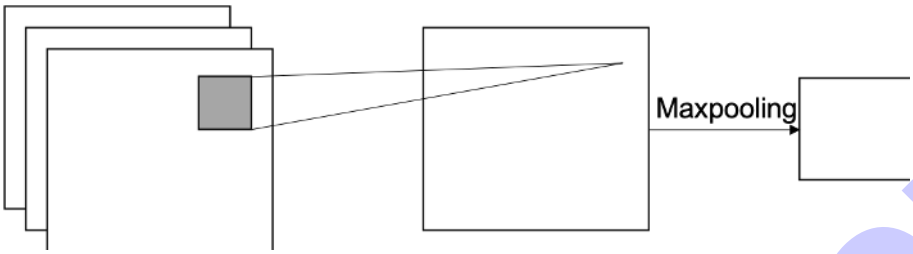
0 1 2

3.6 Instead of taking 64×64 images as input, you now want to be able to train the network to segment images of arbitrary size > 64. List, explicitly, two different approaches that would allow this. Your new network should support varying image sizes in run-time, without having to be re-trained. making it fully convolutional using 1x1 conv layers.

- fully convolutional: 1x1 conv or removing fc1, using FCN, U-Net (1p)
- preprocess: resize, crop, pre-process all the input images to 64x64 (1p)
- ADAPTIVE pooling (Average, Max etc) with fixed output size before the fc1 (1p)
- pooling with fixed output size and does not mention "ADAPTIVE" explicitly (0.5p)
- if approaches too similar (e.g. 1x1 conv + FCN) (1p in total)
- not the intended answer but ok: Graph Neural Network, RNN or LSTM, Transformer
- No: add conv in the beginning, cannot produce fixed size output for variable image size

Problem 5 Backpropagation and Convolutional Layers (12 credits)

Your friend is excited to try out those "Convolutional Layers" you were talking about from your lecture. However, he seems to have some issues and requests your help for some theoretical computations on a toy example.
Consider a neural network with a convolutional (without activation) and a max pooling layer. The convolutional layer has a single filter with kernel size $(1, 1)$, no bias, a stride of 1 and no padding. The filter weights are all initialized to a value of 1. The max pooling layer has a kernel size of $(2, 2)$ with stride 2, and 1 zero-padding.



You are given the following input image of dimensions $(3, 2, 2)$:

$$x = \left(\begin{bmatrix} 1 & -0.5 \\ 2 & -2 \end{bmatrix}, \begin{bmatrix} -2 & 1 \\ -1.5 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \right)$$

0 1 2 a) Compute the forward pass of this input and write down your calculations.

Forward pass

$$\begin{bmatrix} 1 & -0.5 \\ 2 & -2 \end{bmatrix} + \begin{bmatrix} -2 & 1 \\ -1.5 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0.5 \\ 0.5 & -1 \end{bmatrix} \quad (1p)$$

After max pooling,

$$\begin{bmatrix} 0 & 0.5 \\ 0.5 & 0 \end{bmatrix} \quad (1p)$$

0 1 b) Consider the corresponding ground truth,

$$y = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Calculate the binary cross-entropy with respect to the natural logarithm by summing over all output pixels of the forward pass computed in (a). You may assume $\log(0) \approx -10^9$. (Write down the **equation** and keep the logarithm for the final result.)

$$BCEloss = -\sum_i t_i \log s_i \qquad (0.5p \text{ for either his or the line below})$$
$$= -\log(2w_1 - 1.5w_2) - \log(-0.5w_1 + w_2)$$
$$= -\log(0.5) - \log(0.5) = 2\log 2 \qquad (1p)$$

0 ½ c) You don't recall learning the formula for backpropagation through convolutional layers but those 1×1 convolutions seem suspicious. Write down the name of a common layer that is able to produce the same result as the convolutional layer used above.

Fully-connected layer

d) Update the kernel weights accordingly by using gradient descent with a learning rate of 1. (Write down your calculations!)

0 1 2 3 4 5

Partial derivatives for w1/w2 (2p),

$$\frac{\partial BCE}{\partial w_1} = -\frac{\partial \ln(2w_1 - 1.5w_2) + \ln(-0.5w_1 + w_2)}{\partial w_1} = -\frac{2}{2w_1 - 1.5w_2} - \frac{-0.5}{-0.5w_1 + w_2} = -4 + 1 = -3$$

$$\frac{\partial BCE}{\partial w_2} = -\frac{\partial \ln(2w_1 - 1.5w_2) + \ln(-0.5w_1 + w_2)}{\partial w_2} = -\frac{-1.5}{2w_1 - 1.5w_2} - \frac{1}{-0.5w_1 + w_2} = 3 - 2 = 1$$

Update using gradient descent for w1/w2 (2p),

$$w_1^+ = w_1 - lr * \frac{\partial BCE}{\partial w_1} = 1 - 1 \times (-3) = 4$$

$$w_2^+ = w_2 - lr * \frac{\partial BCE}{\partial w_2} = 1 - 1 \times 1 = 0$$

Derivate and update for w3 (1p total):

$$\frac{\partial BCE}{\partial w_3} = 0$$

$$w_3^+ = w_3 - 0 = 1$$

1p if the person only wrote at least the gradient descent update rule

0 1 2

e) After helping your friend debugging, you want to showcase the power of convolutional layers. Deduce what kind of 3×3 convolutional filter was used to generate the output (right) of the grayscale image (left) and write down its 3×3 values.

Vertical edge detector (1p)

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} (1p)$$

Flipping & Scaling are OK

0 1

f) He finally introduces you to his real problem. He wants to find 3×3 black crosses in grayscale images, i.e., each pixel has a value between 0 (black) and 1 (white).

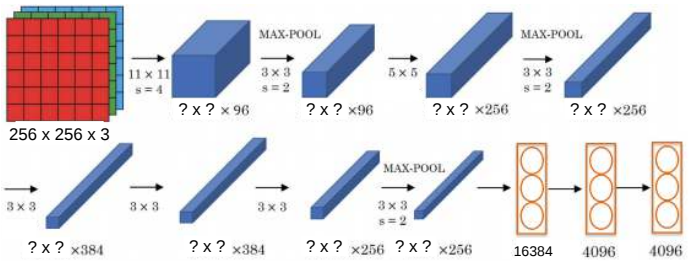
You notice that you can actually hand-craft such a filter. Write down the numerical values of a 3×3 filter that maximally highlights on the position of black crosses.

$$\begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix} (2p)$$

Flipping & Scaling are OK, even though pixel values were given

Problem 7 Convolutional Neural Networks (7.5 credits)

Your friend needs your expertise in classifying a set of RGB Images of size 256 x 256 pixels into a total of 1000 classes. Can you help him out?
Since you learned that CNNs are great to tackle these sort of tasks, you decide to start out with the following CNN architecture.



Notes:

- The values directly below the arrows indicate the filter sizes f of the corresponding convolution and pooling operations.
- s stands for stride. If no stride is specified, a stride of $s = 1$ is used.
- All convolutional and pooling layers use a padding of $p = \frac{f-1}{2}$ for a corresponding filter size of f.
- For each convolutional filter, we include a bias.

0 1 a) In the figure above, the output layers for classification are missing. Explain:

1. Which type of last layer you would use there and how would you choose it's dimension?
2. Which output function and loss function would you choose for this task?

1. Fully-connected layer (0.5 points) with 1000 neurons (0.5 points)
2. Softmax function combined with Cross-Entropy Loss (0.5 point)

0 1 b) Calculate the output dimension of the image after passing through the first convolutional layer. Make sure to include your calculations in the solution.

Formula to calculate the output dimension: $\dim_{out} = \frac{\dim_{in} - f + 2 \cdot p}{s} + 1$ Output Dimension $(\frac{256-11}{4} + 1) = 64 \times 64 \times 96$ (1 Point)

0 1 c) Calculate the number of learnable parameters in the first convolutional layer. Make sure to include your calculations in the solution. Un-multiplied answers are accepted (e.g., $3 * 3 * 16$)

Number of Parameters $(11 * 11 * 3 + 1) * 96 = 34944$ parameters (1 Point)

d) Calculate the size of the combined receptive field of the first two and of the first three layers. This corresponds resp. to the area of pixels in the input image that each neuron

0 1 2

1. after the second layer (right after the first MAX-POOL layer)
2. after the third layer (before the second MAX-POOL layer)

"sees".
Hint: Strides affect the total receptive field sizes of subsequent layers.

Size of total receptive field after $k > 1$: $r_k = r_{k-1} + \left(\prod_{i=1}^{k-1} s_i\right) \cdot (f_k - 1)$

- layer 1: $r_1 = 11$ (11×11 filter)
- layer 2: $r_2 = 11 + (4 \cdot (3 - 1)) = 11 + 8 = 19$ (0.5p answer, 0.5p calculation)
- layer 3: $r_3 = 19 + (4 \cdot 2 \cdot (5 - 1)) = 19 + 8 \cdot 4 = 19 + 32 = 51$ (0.5p answer, 0.5p calculation)

Alternative solution for r_3 (from right to left, needs separate calculation for r_2!): $r_{i-j} = f_j + (r_{i-(j+1)} - 1) * s_j$)

- 3rd to 2nd layer: $r_{3-2} = 5$ (5×5 filter)
- 3rd to 1st layer: $r_{3-1} = 3 + (5 - 1) \cdot 2 = 11$ (3×3 filter, stride 2)
- 3rd layer to input: $r_3 = r_{3-0} = 11 + (11 - 1) \cdot 4 = 51$ (11×11 filter, stride 4)

After a series of convolutional layers, the architecture has 3 fully connected layers that help process the spatial information obtained by the convolutions and prepare it for the output layer. Our friend just found an article about image segmentation and gets really excited about this. He decides to forget about the classification task and wants to work on image segmentation.

We don't need to reject our architecture: We first convert our architecture to a fully-convolutional network by ditching the fully connected layers. Next, we want to produce an output similar to the input size from this bottleneck where we would like to mirror the current architecture.

e) How would you replace the convolutional layers in the mirrored architecture to increase the the image size from our bottleneck onwards?

0 1

Accepted answers: upsampling/unpooling + convolution, transposed convolution.

Grading notes: upsampling alone (without conv) is 0.5p, "inverse convolution", "transformed convolution", and all other misspellings are 0.5p.

f) The new architecture is able to process an image of any input size. How about the original architecture that we started with, was it able to handle images of arbitrary sizes as input, too? Give an explanation for your answer.

0 1

No, the original architecture was not able to handle images of arbitrary size (0.5p).

Explanation: Fully-connected layers require fixed size and cannot handle variable output size of convolutional layers (0.5p).