Ex7 - Deep Learning Questions - Computer Vision - school solution

– please listen to the pre-recorded video as well–

1. Explain each of these terms in a sentence or two:
    1. Cross entropy
    2. Residual connections
    3. Adam optimizer
    4. Cyclic learning rate
    5. Dropout
    6. Bottleneck layer
    7. 1x1 convolution
    8. DenseNet

2. Explain the pros and cons of using small and large batch sizes.

Large batch size - **cons:** in the extreme case, where we take into account all the dataset, we can be stuck on a local minimum because averaging all the gradient directions might cancel out each other.

Small batch size - **cons:** if we take again the extreme cases, where we consider only one datapoint, the gradient can be not accurate enough.

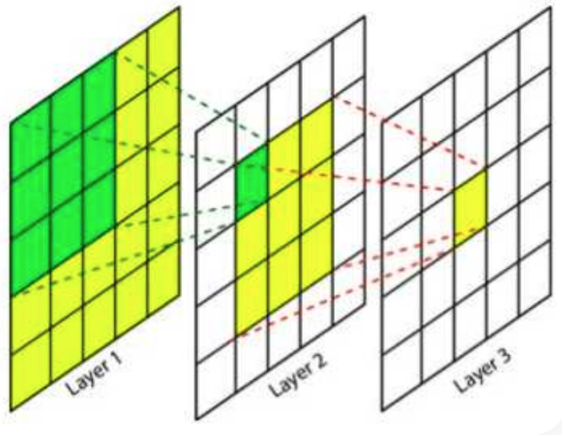
Therefore, using a large batch size is a good thing in order to have accurate gradients but should not be too large.

3. How many 3x3 filters are needed to replace a 7x7 kernel? Compare the number of parameters in each option.

When we use a 7x7 kernel the central pixel is a weighted sum of 0 to 49 pixels.

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	0	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48

Let’s have a look at the figure below, that illustrates the receptive field of two consecutive convolutional layers. Consider a 3x3 kernel, the output pixel (layer 3) is influenced by 3x3 neighbors from layer 2. But,the pixels in layer 2 are influenced by a previous convolutional layer. So at the end, the pixel at layer 3 is influenced by 5x5 if we take into account the two consecutive convolutional layers.



Let’s go back to our question: how many 3x3 kernels have the same influence as this 7x7 kernel over an output pixel?
1 kernel: 3x3 pixels influence the output value - green frame
2 kernels: 5x5 pixels influence the output value - orange frame
3 kernels: 7x7 pixels influence the output value - blue frame

3 kernels of 3x3 have: 3x3x3=27 parameters
1 kernel of 7x7 have: 49 parameters

4. You are training a neural network for classifying images on a custom dataset and it doesn't seem to learn anything. Describe your approach to solving the issue.
In the best practices lecture we discuss this problem a lot - there is NO one answer to this question.
I would start by simplifying as much as I can - working on just one image, a very simple network etc…
Moreover, visualizing the dataset, the network output and changing some training arguments (lr, optimizer) might help to understand where the problem is.

5. Mention the problems imbalanced datasets can cause to Deep Learning problems, and suggest a few ways to avoid them.
An Imbalanced dataset can cause a biased network. The network will be trained better in the class that has more data points.
We can detect them by visualizing a histogram of data points in each class. We can also print or visualize the matrix confusion as we saw in the first exercise.

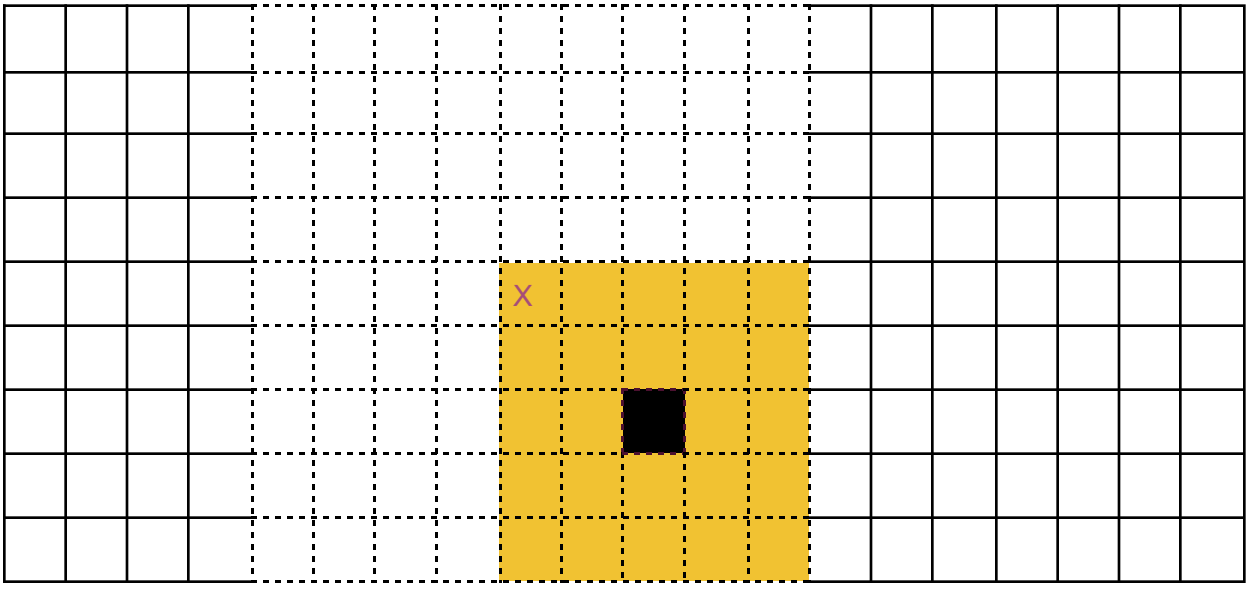
In order to avoid it we need to balance the dataset. This can be done by considering augmentations over the poor classes or getting rid of examples from the rich classes. In case that the imbalancing is naturally “good” (think for example of spam and not spam binary classification - one might have much more non-spam emails than spam), we can consider balancing this by a dedicated loss that multiplies the poor class by a bigger factor to compensate its smaller dataset.

6. Given two networks: SRCNN (https://arxiv.org/pdf/1501.00092.pdf) and Unet (https://arxiv.org/pdf/1505.04597.pdf) . What is the receptive field of a pixel in each network? You should consider one pixel (let’s say the center pixel of the output) and go back to see what neighbors at the input image influence this pixel.

The receptive field is influenced only from the spatial dimension of the kernel. The number of kernels (the third dimension) has nothing to do with the receptive field calculation.

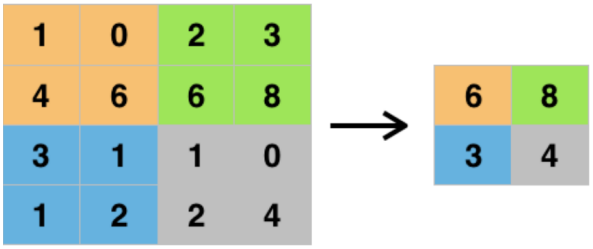
(i) SRCNN - considering $f_1,f_2,f_3 = 9,1,5$, no downscale or strides between the layers. A pixel in the last third (last) layer is a result of a 5x5 convolution from the second layer. The second layer pixels are a result of a 1x1 convolution (“no receptive field”, every pixel is mapped to a combination of the channels without any spatial information). The pixels of the second layer are the result of 9x9 convolution. To sum up, a pixel at the very last layer has an influence of 13x13 pixels from the input image (4 pixels at every direction from the first convolutional layer, 2 pixels at every direction from the third convolutional layer -> 6 pixels at every direction in total).

Black - output pixel
Yellow - considering 5x5 kernel
Dashed - considering the X pixel - is a result of the weighted sum of the dashed.
In total, the black pixel is influenced by 6 pixels in each direction.

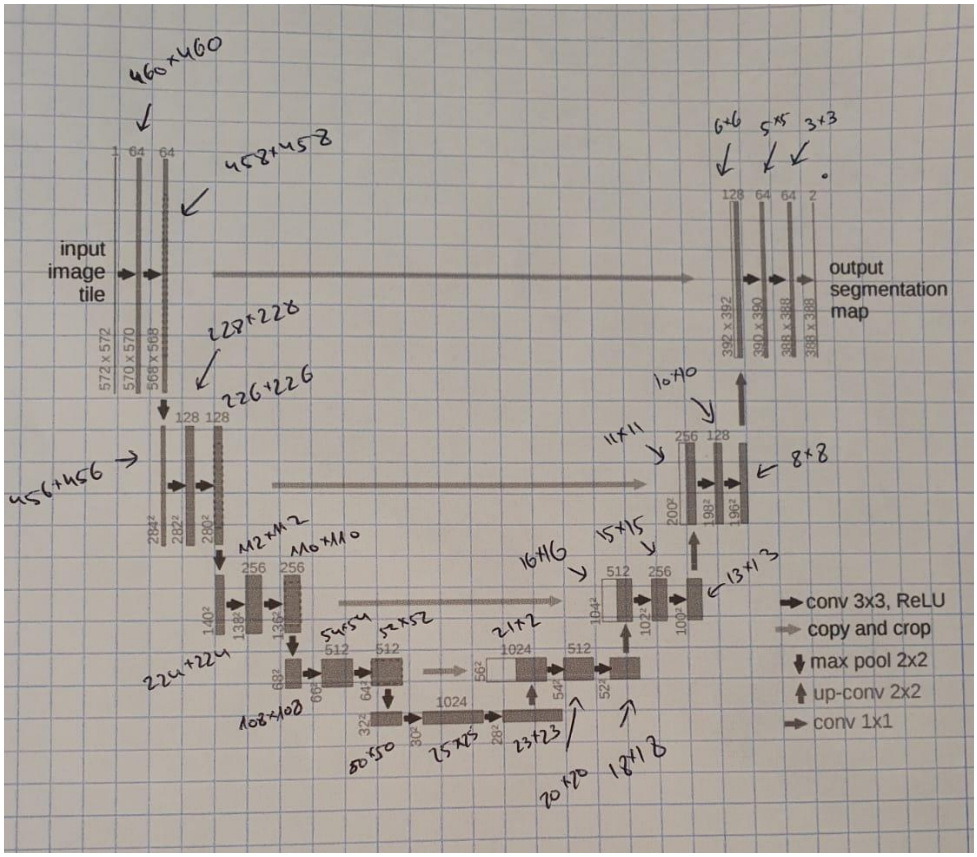


considering f_1,f_2,f_3 = 9,5,5 no downscale or strides between the layers (as was in the SR exercise). The same goes here. considering the last convolutional layer - we have an influence of two pixels in every direction. We need to add two more pixels from the second convolutional layer and 4 more due to the first convolutional layer -> 8 in total which means that the respective field is 17x17 in this case.

(ii) UNET. The receptive field in UNET architecture is influenced from the convolutional layers and mostly from the max-pooling layer. As we saw previously in this exercise, every convolutional layer contributes to the output pixel by half a kernel in every dimension.
A max-pooling layer doubles the receptive field (if we apply max-pooling by a factor of 2). Why?



Every pixel in the right image is influenced by 4 pixels in the left image. The 2x2 right image is influenced by the receptive field of 4x4.
With this in mind, we will go through all the UNET layers from the output back to the input.



The bottom line of this question is that UNET has a very very large receptive field because of its pooling architecture.

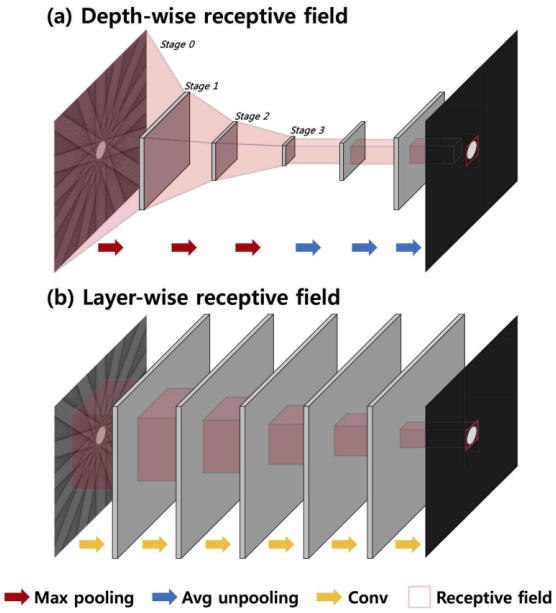


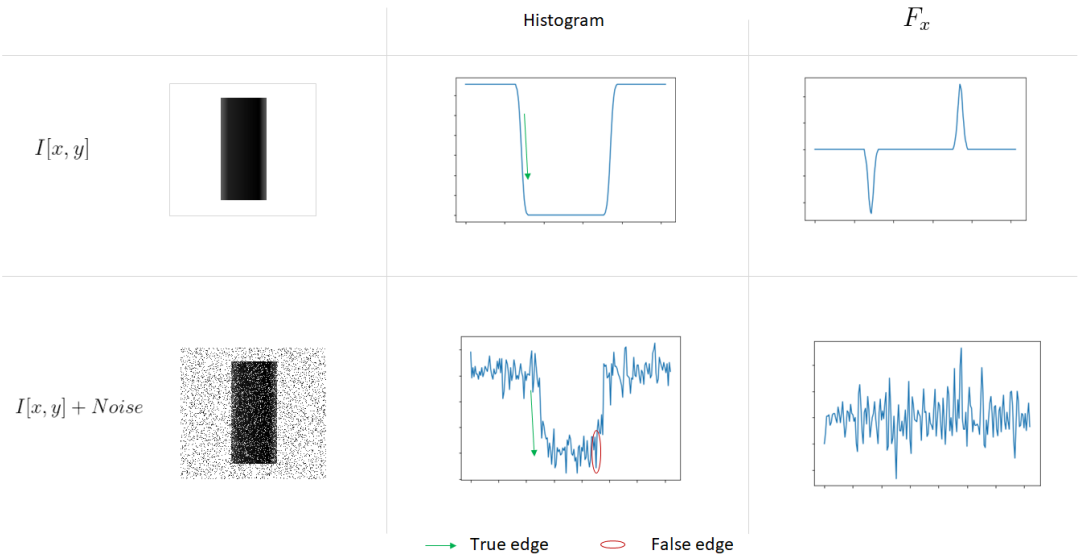
Figure 5. Effective receptive field comparison.

7. Given a standard CNN consists of convolutional layers and then fully-connected layers. Explain what layer in CNN can reach a lot of parameters? How can we avoid it?
The fully connected layer is the layer that is responsible for the big amount of parameters. The number of parameters are in_neuronsXout_neurons. In order to reduce the number of parameters, we can either use less out_neurons or somehow reduce the length of in_neuron. In_neuron is usually a result of few convolutional layers and pooling (MxMxC) - so we can downscale the image more (to have smaller MxM) or use fewer filters (smaller C).

8. What is the result of convolving an image X with a filter h = [-1 -1 -1; 0 0 0; 1 1 1].
This is a special kernel. In order to understand what this kernel does I would analyze each dimension separately.
-1 -1 - 1
0 0 0
1 1 1
If we consider the horizontal direction, we have negative and positive numbers [-1 0 1] and this is exactly the vertical edge detector (derivative). However, if we consider a vertical line in this kernel (i.e., [1 1 1]) this is a summation which can be considered as blurring in the opposite direction of the derivative. To sum up, this kernel blurs in one direction and calculates the derivative in the opposite direction.

BTW: why should we use this type of kernel?

The derivative of an image results in amplifying the noise in an image. If we would like to analyze the derivative image it would be very hard (see figure below). Blurring operators make the image smoother and also get rid of the noise (a bit). So, usually, before applying the derivative, we should smooth the image. The kernel does both operations at once.



9. (a) What if all the weights are initialized with the same value?

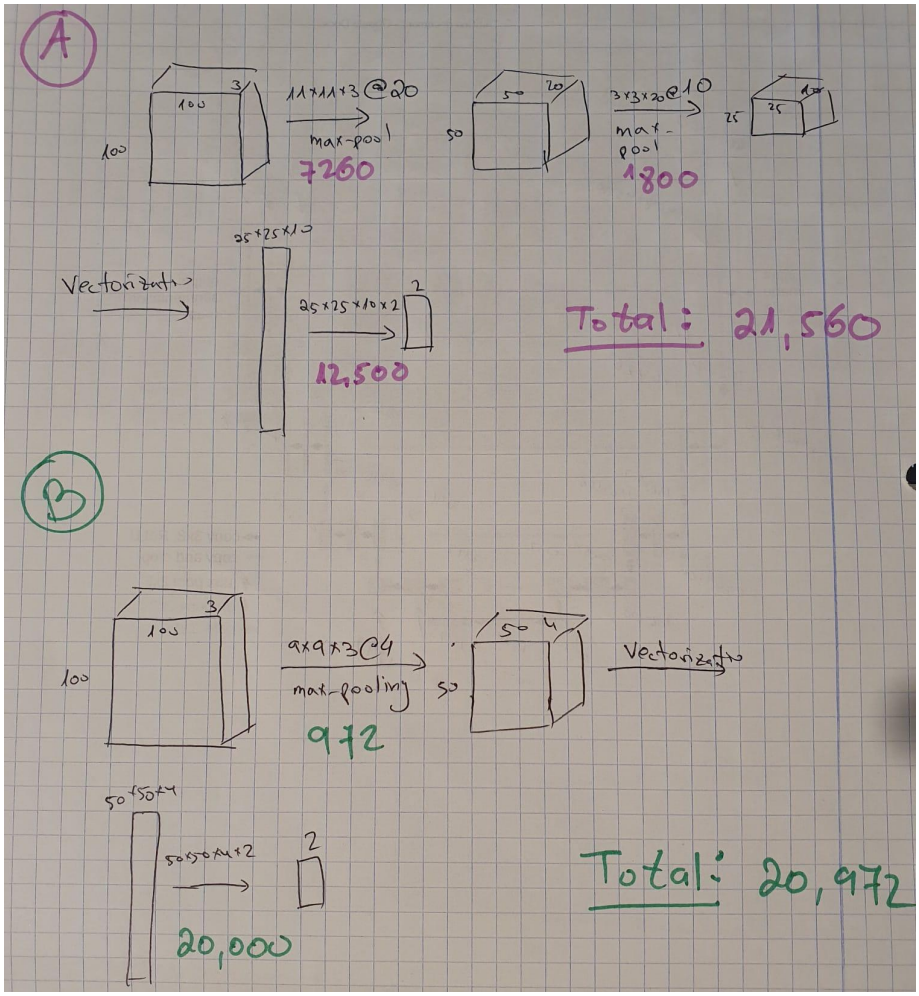
The network won't learn anything.
Think about a back propagation step of weights in a last layer. The weights are updated by calculating the derivative of the loss in respect to these weights. The derivatives will be the same, hence the updated weights (that were with the same value before the update) will be the same value after the update step. So, there is no learning in this process. All the weights are changing to the same values.

(b) What happens if we set all the biases to be zero (ignoring the biases)?

One can read this question in two different ways:
(i) We can set the biases to be zero at the beginning of the training process and they will be changed during the training.

(ii) We can get rid of the biases and the network will still work and be trained, the network will be less complex and flexible though. You can read more in https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98 (under 'Bias as a single neuron' section).

10. Write down two NN models (it can also be convolutional networks) that have (more or less) the same number of parameters, but with different power of computational. Explain your answer.



The two questions below are open questions. We would like to see the way you develop algorithms using deep learning. Your answers should be written (not code). You can use pseudo code if it is more convenient for you.

11. You moved into a new apartment and you would like to make your front door smarter. Write an algorithm that alerts every time there is someone outside your door who is not a part of your family (you decide who is considered to be family).

"Training phase"

- Put a camera in your front door
- Use a pre-trained network, A, that has a good representation (feature extractor) like the siamese network with a constructive loss or triplet loss (see Face detection lecture in the foundation track).
- Take a few photos (I would say 20) of all the people you would like to detect as family members
- Feed the images of every family member (identity) into the network A.
- Calculate clusters for every identity. This step gives us the average feature vector and the radius for every identity.

At running time ("inference phase"):

- Take a photo from the camera of your front door every few seconds
- Analyze this frame by detecting all the faces using an off-the-shelf face detection algorithm
- Feed every detected face into the network A to have a feature vector X
- Go over all the identity average features, Y with its radius R:
    - Calculate the distance between X and Y
    - If the distance is more than the R, try the next identity Y
    - Else, return that this is a family manner and continue to the next face X in the frame

More options to solve this problem (and there are much more…):

(1) Classification:
    - Gather more images of your family and train a neural network to classify them.
    - Track over the probabilities that each class has
    - For every new face from the front camera, analyze the probability vector. Hopefully, if this is not a family member the probability for all the classes will be low.

(2) The clustering in the "training phase" is for having a kind of thresholding values. We can also use a different approach for having a threshold: for example, by calculating the distances between all the pairs of the same identity and deciding that the largest distance is the threshold.

Note: For all the solutions above, it is better to gather the photos of your family from the camera you put in your front door so the images from the “training phase” and the “inference phase” will have the same resolution, colors, angle…
You can also add photos with and without light from the hallway, use props (sunglasses, hats…) to enlarge the dataset and make our algorithm more robust.

12. These days, most of the parking lots have a system that recognizes the license plate number of an entering car. This helps to automatically open the gate in case that the driver paid before arriving at the exit gate. For this purpose, two cameras are needed - one at the entrance and one at the exit. Both capture the license plate number and translate it into a series of numbers.

Write an algorithm that for a given image containing a car with a license plate number, recognizes all the numbers in this plate. Pay attention that the angle and the location of the license plate can vary between images.

For simplicity: all the license plates have the same size (in the real world, not in the image), have 7 digits and have the same font. The plate is yellow with black digits. Some pictures are attached.
Moreover, no trucks or motorcycles can enter the parking lot.
If you have more assumptions, please write them down.

In my solution I first detect the license plate in the image, then, detect the bounding box of each digit and then recognize it.

“Training phase” - once we have a bounding box of a digit we would like to classify it.

1) Try mnist
2) Create your own dataset (this is the same font so it is an easy task) and train a network to classify them.

“Inference phase” -

Detect the license plate.
In order to detect the license plate we can do one of the following:

1) Analyze only the “yellow” channel of the image (all the pixels that are in the red and green channels above some threshold) -> then take the bounding box of these pixels.
2) Take a yellow box in a reasonable size and apply template matching for several scales-> then take the location with the highest value after non-maximum suppression
3) Annotate several hundreds of images with bounding boxes around the license plate and train a network.

Detect the digits bounding box and Feed every bounding box into the classification network
If we explore the dataset, we see that the digits are not overlapping so we can find the spaces between them in order to detect the bounding box of each digit.

1) Put a flag that we are in “out-digit” (not inside a digit)
2) Go over vertical lines in the license plate bounding box (x axis)
3) If the sum is non-zero:
    a) If flag == “out-digit” -> change the flag to “in-digit” and save the location of the x axis in X1
    b) Else (means that we are in digit) -> continue (2)
4) If the sum is zero:
    a) If flag == “in-digit” -> change it to “out-digit” and take the bounding box from X1 to current x axis. Feed the network with this small bounding box containing the digit.
    b) Else -> continue (2)

Assumptions:

- No yellow cars are getting into the parking lot.

Note:
One can apply template matching of all the digits with the image to find them. I guess that there are no digits that are not related to the car in this part of the frame.