Very Deep Learning
Assignment 3
Group No. 8

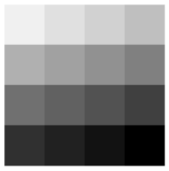
1 Semantic Segmentation via Fully Convolutional Networks

1.1 Theory problem

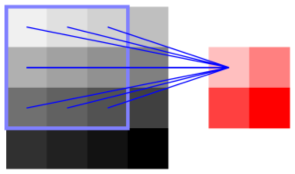
1. How a fully connected layer can be realized as a convolution layer?

Consider a grayscale picture, data in a picture can be represented by numbers as it shows here.

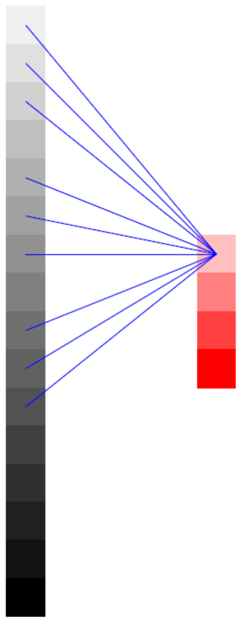
$$\begin{bmatrix} 0.0625 & 0.125 & 0.1875 & 0.25 \\ 0.3125 & 0.375 & 0.4375 & 0.5 \\ 0.5625 & 0.625 & 0.6875 & 0.75 \\ 0.8125 & 0.875 & 0.9375 & 1 \end{bmatrix} \rightarrow$$



A convolution filter applies on an area as it shows below.

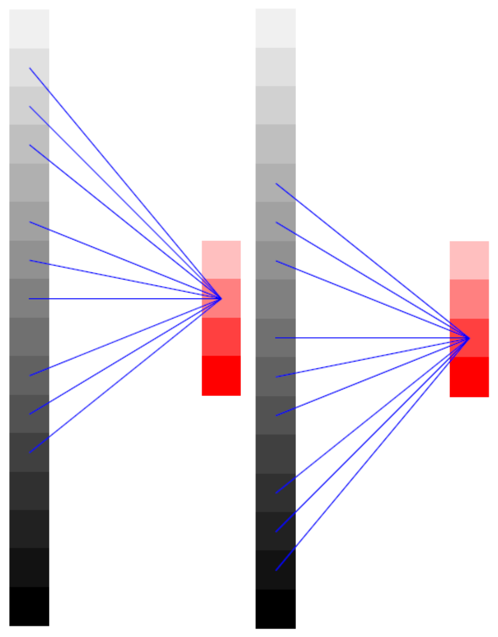


The blue square is an area where the convolution applies. A result will be kept in a red neuron on the right. It is possible to rearrange the data (Which are usually tensors) into vectors. The next picture is the same objects as the previous picture shows but rearranged.



Now, the picture looks like two layers of neurons. The blue lines represent weights that connect a red neuron with input neurons. Mathematically, this red neuron also connected with the other input neurons (The rest of the input neurons) but weights of this connection must be zeros to realize convolution network, therefore, it is not necessary to represent them with lines.

If we shift the filter one stripe to the right, the connection to input neuron will shift by 1 which results in the left picture. If we shift the filter one stripe to the south, the connection to input neuron will shift by the width of the picture which results in the right picture.



This is how we realize convolutions network as a fully connected network. It is also possible to write down a matrix that connects input and the next layer. However, in the training process, these connection weights must be the same to realize convolution filter

2. What is the importance of skip connections in a CNN for image segmentation and object detection problems?

Sometimes information processing might destroy some important information. Skip connections allow the important information to survive the process to be used in another process.

3. Ground truth labels for image classification problem are class names which are converted to one hot vectors and cross-entropy loss is applied over CNN to train them in supervised manner. What are the ground truth labels in semantic segmentation problem and by what loss function are they trained over a CNN?

Ground truth is a set of information made by humans. We use ground truths to train the networks. They are answers of the image segmentation problem. To make one of them, a human must segment an image by their self.

4. How transposed convolution helps in up-sampling an image, in case of semantic segmentation?

A transposed convolution is a convolution that result in an output which bigger than the input. This convolution could be anything. The process itself results in up-sampling. We need to define the up-sampling operation as a layer in the network in order to perform training using backpropogation where the image and respective segmentation ground truth will be given to us.

5. Why accuracy is not a good measure in case of semantic segmentation? What measure is used to evaluate results in semantic segmentation?

The pixel accuracy is commonly reported for each class separately as well as globally across all classes.

$$\text{accuracy}= \frac{TP+TN}{TP+TN+FP+FN}$$

This metric can sometimes provide misleading results in case of semantic segmentation when the class representation is small within the image, as the measure will be biased in mainly reporting how well you identify negative case (ie. where the class is not present).

The Intersection over Union (IoU) metric, also referred to as the Jaccard index, is often used for evaluation of semantic segmentation. It measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across both masks.

$$\text{IoU}= \frac{\text{target} \cap \text{prediction}}{\text{target} \cup \text{prediction}}$$

The IoU score is calculated for each class separately and then averaged over all classes to provide a global mean IoU score of our semantic segmentation prediction.

2 Visualizing Convolution Neural Networks

2.1 Theory Problems

1. What is a receptive field of a convolution filter?

The receptive field in Convolutional Neural Networks (CNN) is the region of the input space that affects a particular unit of the network. A receptive field of a feature can be described by its center location and its size. When the receptive field term is mentioned, it is taking into consideration the final output unit of the network (i.e. a single unit on a binary classification task) in relation to the network input (i.e. input image of the network).

2. How can a receptive field of a filter of a particular layer be obtained in theory?

Receptive field in each layer can be calculated by using the following equations:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

$$j_{out} = j_{in} * s$$

$$r_{out} = r_{in} + (k-1) * j_{in}$$

$$start_{out} = start_{in} + \left(\frac{\mathrm{k} - 1}{2} - \mathrm{p} \right) * j_{in}$$

$n_{in:}$ number of input features

p: convolutional padding size

$n_{out:}$ number of output features

s: convolutional stride size

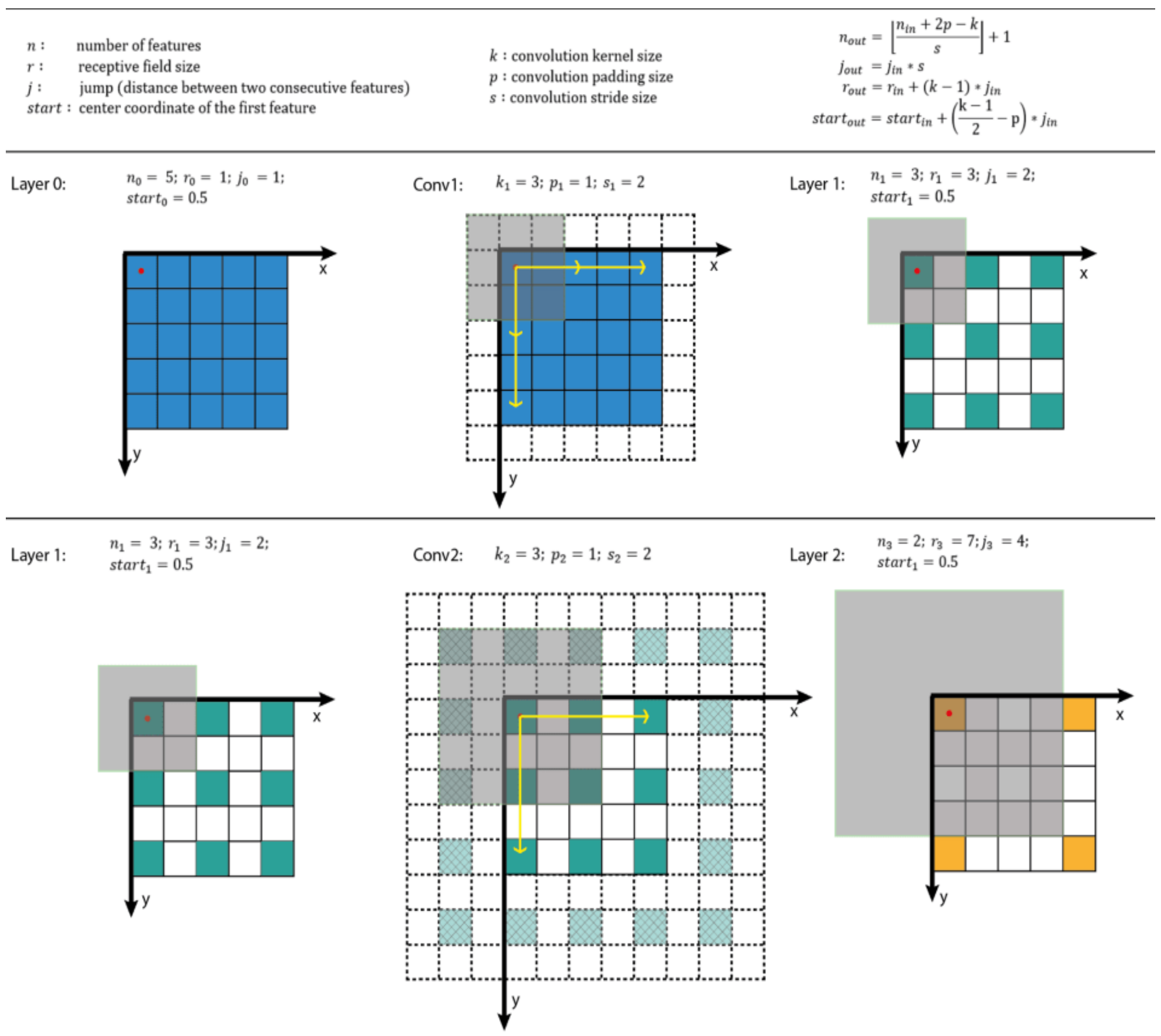
k: convolutional kernel size

j_{in}/j_{out}:distance between two adjacent features

$r_{out:}$ size of receptive field

$start_{out}$: centre coordinate of one pixel

The following figure shows the details of re programmed in python to provide the n fea layer.



n : number of features
r : receptive field size
j : jump (distance between two consecutive features)
$start$: center coordinate of the first feature
k : convolution kernel size
p : convolution padding size
s : convolution stride size
$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$
$j_{out} = j_{in} * s$
$r_{out} = r_{in} + (k-1) * j_{in}$
$start_{out} = start_{in} + \left(\frac{k-1}{2} - p \right) * j_{in}$
Layer 0: $n_0 = 5;\ r_0 = 1;\ j_0 = 1;$ $start_0 = 0.5$
Conv1: $k_1 = 3;\ p_1 = 1;\ s_1 = 2$
Layer 1: $n_1 = 3;\ r_1 = 3;\ j_1 = 2;$ $start_1 = 0.5$
Layer 1: $n_1 = 3;\ r_1 = 3;\ j_1 = 2;$ $start_1 = 0.5$
Conv2: $k_2 = 3;\ p_2 = 1;\ s_2 = 2$
Layer 2: $n_3 = 2;\ r_3 = 7;\ j_3 = 4;$ $start_1 = 0.5$
x
y

3. How can a receptive field of a filter be obtained in practice?

Receptive field is a way of measuring network components dependency to calculate this we need a set of parameters for each layer: filter_size kk, stride ss, offset (padding) pp, and calculate the compound parameter when layers are stacked together.

To calculate the combination of two layers: k_0,s_0 and k_1,s_1, we can use:

$$i_0=s_0(i_1-1)+p_0\pm\frac{(k0-1)}{2}$$

$$i_1=s_1(i_2-1)+p_0\pm\frac{(k1-1)}{2}$$

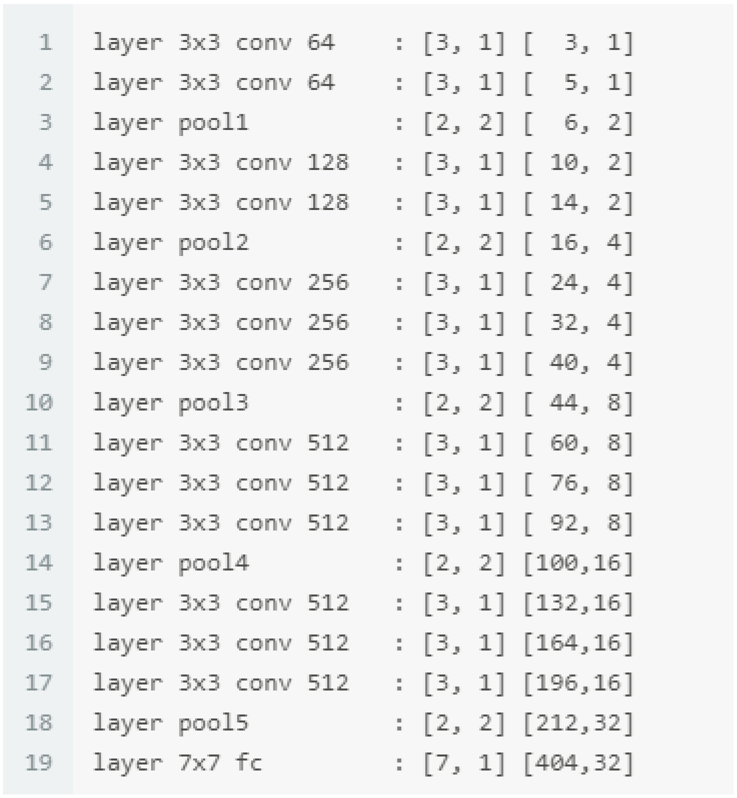
A simple example is displayed below:

```
vgg_16 = [
            # 1
            [3, 1], [3, 1], [2, 2],
            # 2
            [3, 1],  [3, 1], [2, 2],
            # 3
            [3, 1], [3, 1], [3, 1], [2, 2],
            # 4
            [3, 1], [3, 1], [3, 1], [2, 2],
            # 5
            [3, 1], [3, 1], [3, 1], [2, 2],
            # fc6, fake convolutional layer
            [7, 1]
]
vgg16_layers = [
            "3x3 conv 64", "3x3 conv 64", "pool1",
            "3x3 conv 128", "3x3 conv 128", "pool2",
            "3x3 conv 256", "3x3 conv 256", "3x3 conv 256", "pool3",
            "3x3 conv 512", "3x3 conv 512", "3x3 conv 512", "pool4",
            "3x3 conv 512", "3x3 conv 512", "3x3 conv 512", "pool5",
            "7x7 fc"
]
def cal_receptive_field(kspairs, layers=None):
            # K: composed kernel, also the receptive field
            # S: composed stride
            K, S = 1, 1
            # H = 224
            if not layers:
                        layers = range(len(kspairs))
            for layer, kspair in zip(layers, kspairs):
                        k, s = kspair
                        K = (k-1) * S + K
                        S = S * s
                        # H = H//s
                        # iamge size {0}'.format(H)

                        print('layer {:<15}: {} [{:3},{:2}]'.format(layer, kspair, K, S))

cal_receptive_field(vgg_16, vgg16_layers)
```

Ouput:



```
layer 3x3 conv 64     : [3, 1] [  3, 1]
layer 3x3 conv 64     : [3, 1] [  5, 1]
layer pool1           : [2, 2] [  6, 2]
layer 3x3 conv 128    : [3, 1] [ 10, 2]
layer 3x3 conv 128    : [3, 1] [ 14, 2]
layer pool2           : [2, 2] [ 16, 4]
layer 3x3 conv 256    : [3, 1] [ 24, 4]
layer 3x3 conv 256    : [3, 1] [ 32, 4]
layer 3x3 conv 256    : [3, 1] [ 40, 4]
layer pool3           : [2, 2] [ 44, 8]
layer 3x3 conv 512    : [3, 1] [ 60, 8]
layer 3x3 conv 512    : [3, 1] [ 76, 8]
layer 3x3 conv 512    : [3, 1] [ 92, 8]
layer pool4           : [2, 2] [100,16]
layer 3x3 conv 512    : [3, 1] [132,16]
layer 3x3 conv 512    : [3, 1] [164,16]
layer 3x3 conv 512    : [3, 1] [196,16]
layer pool5           : [2, 2] [212,32]
layer 7x7 fc          : [7, 1] [404,32]
```

1.2 Practical Problem

Implemented.

2.2 Practical Problem

Unable to access GPU.