Advanced Computer Vision

Exercise Sheet 6

Winter Term 2023
Prof. Dr. Niels Landwehr
Dr. Ujjwal

Available: 12.12.2023
Hand in until: 19.12.2023 23:59
Exercise session: 22.12.2023

Task 1 – Saliency Map in Python [15 points]

In this task, you implement the computation of a saliency map in Python. As explained in the lecture, the saliency map is obtained by computing for a given model and image the gradient of the score of the predicted class with respect to the model input, at the point of the input image.

Write a Python notebook that computes the saliency map for an image. Specifically, in your notebook

1. Load a model pretrained on ImageNet (recommendation: ResNet50, see `https://keras.io/api/applications/`).
2. Compute the prediction of the model on the example image *ouzel.png* and verify that it predicts the correct class ("ouzel" is class index 20 within ImageNet). Make sure to preprocess the image correctly (`https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet/preprocess_input`).
3. Obtain the gradient of the class score for the correct class index with respect to the input image. For Tensorflow, we provide a function *gradient_input.py* that you can use as building block in your notebook if you want. For this exercise, it is ok to compute the gradient of the class probability with respect to the input rather than the gradient of the class score with respect to the input (pretrained models usually include softmax in the last layer).
4. Construct a saliency map from the obtained gradient. As explained in the lecture, take the absolute value of the gradient and maximize over color channels. Save your saliency map as a greyscale image and compare it to the input image.

Task 2 – Gradient of Node at Intermediate Layer With Respect to Input [20 points]

In this exercise, we further study gradients with respect to the input, but for nodes in intermediate layers. In the provided notebook *gradient_inner_node.ipynb*, we define a simple convolutional neural network model that takes an input of size 128×128 and processes it with several stacked convolution layers $L_1, ..., L_6$, some of them with a stride of two to reduce the spatial dimension. The network then produces class probabilities for ten classes.

For a convolution layer $L_k \in \{L_1, ..., L_6\}$ in the model, let $z^{(k)}_{x,y,c}$ denote a single element at spatial position $x, y \in \mathbb{N}$ and channel $c \in \mathbb{N}$ of the output of the layer L_k. Write a function that computes the gradient

$$\frac{\partial z^{(k)}_{x,y,c}}{\partial \mathbf{x}} \in \mathbb{R}^{128\times 128}$$

of the output element $z^{(k)}_{x,y,c}$ with respect to the network input $\mathbf{x} \in \mathbb{R}^{128\times128}$.

Visualize a few of these gradients (for different layers) by converting them to maps as we did in Task 1. For the visualizations, use the initial model (without training) and random inputs, drawing each input position uniformly between zero and one. You will notice that the entries in the gradient are non-zero only for a rectangular subset of pixels in the 128×128 input (consider a nonlinear color scale when visualizing to be able to better differentiate between zero and small non-zero values). Only these pixels are "seen" by the node $z^{(k)}_{x,y,c}$, that is, only their values influence the value of $z^{(k)}_{x,y,c}$. The size and position of this subset depends on the layer and location of $z^{(k)}_{x,y,c}$, and is called the *receptive field* of the node.

We provide as example for how the solution should look like a visualization of the gradient of the node at spatial position (0,0) and channel 0 in layer L6 as *gradient_L6.png*.

Hint: In Keras, you can easily define a partial model that outputs the activation of an intermediate node of the original model using the functional API for building models (see `https://keras.io/guides/functional_api/`). If you define such a partial model, you can reuse the function of Task 1 to compute the gradient.

Task 3 – Size of Receptive Fields in Convolutional Neural Networks [15 points]

When you compute and visualize gradients of inner nodes in different layers L_k of the model given above, you will have noticed that the size of the receptive field is larger for nodes in layers higher up in the network. Assume a convolutional neural network that processs an input $\mathbf{x}$ by a stack of m convolution layers with strides $s_1, ..., s_m$ and kernel sizes $k_1, ..., k_m$. For example, the model of Task 1 would have $m = 6$, $(s_1, ..., s_m) = (1, 2, 1, 2, 1, 2)$, and $(k_1, ..., k_m) = (5, 5, 3, 3, 3, 3)$.

Find a formula for computing the size of the receptive field for a node at layer m, which can indeed be computed from the $s_1, ..., s_m$ and $k_1, ..., k_m$. Validate your formula by looking at example gradients from Task 2.