Cognitive Computer Vision: Exercise 4

1 CNN for eye fixation prediction

As illustrated in Figure 1, given an input image, an eye fixation prediction system attempts to produce a heatmap that indicates where a human would be likely to pay attention. Many popular systems use the approach of combining features from multiple layers in a CNN, and then creating an fixation map prediction based on all of them. Figure 2 shows an example of such a system, containing several submodules:

- a base CNN shown in the top part consists of convolutional and pooling layers,
- a multi-level feature representation is formed by concatenating features from several layers in the base CNN, and
- a prediction CNN in the lower right part that learns how to combine the different level features, and finally
- sometimes a learned prior is added to account for known biases such as human tendency to focus on the center part of images.

The idea behind using features from multiple levels is that lower level features maintain more information about the spatial location of items and low-level features such as edges in the image, while high level features often have a richer, semantic-level representation corresponding, e.g., to objects.

1.1 Base CNN

We begin by setting up the base CNN shown in the top left part of Figure 2. Define the base CNN as shown in Figure 3. Use similar methods as in the previous exercise, see also the example solution to help you. Define the input images via a placeholder of shape $\left[\texttt{None},\ 224,\ 224,\ 3\right]$. Use `padding="same"` in both the convolutional and max pooling layers.

1.2 Multi-level feature representation

Concatenate the activations from the layers POOL2, POOL3, and CONV8 into a new tensor called `multilevel_features`. Use `tf.concat` and concatenate along the last dimension that corresponds to the number of channels.

- What are the shapes of the activation of POOL2, POOL3, and CONV8? What is then the shape of `multilevel_features`?

1.3 Fixation prediction

The multi-level features can now be used to define a few more layers to predict an eye fixation map. Due to the pooling layers, the multileve feature maps are smaller than the original image. To produce a fixation map of the same size as the original image, we must upsample it. Add layers as indicated in Figure 4. Note that here we ignore adding the learnable prior that is included in Figure 2.

Figure 1: Example of an input image (left) and the corresponding eye fixation map (right). Brighter colour on the right corresponds to higher fixation likelihood.

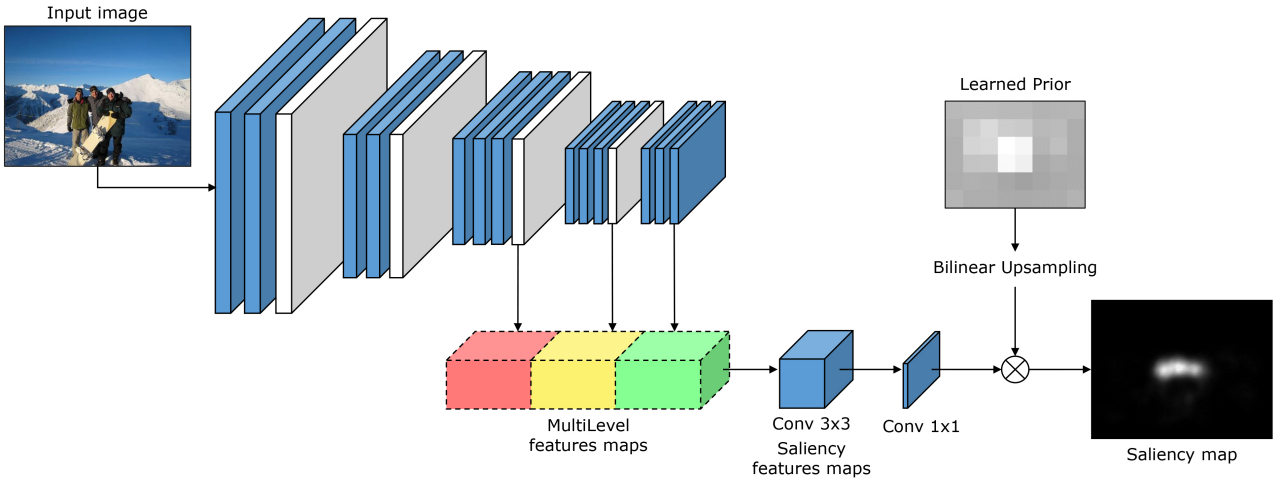

Figure 2: Features from many layers in a CNN are concatenated into a multilevel feature map. Additional convolutional layers are inserted to predict a saliency map or fixation map. Blue layers indicate convolutional layers. Gray layers indicate pooling layers. Image from authors of [1].

- If you do not understand how the 1-by-1 convolution works, see the video `https://www.youtube.com/watch?v=9EZVpLTPGz8` for a concise, intuitive explanation of its usefulness.
- Use `tf.image.resize_images` to do the upsampling with bicubic interpolation.

1.4 Loss function

To begin training the network, we must define a loss function for it. In the following, let $\phi(x)$ be the network's predicted eye fixation map for input image x, and let y be the ground truth eye fixation map. Furthermore, $\phi(x_i)$ and y_i denote the prediction and ground truth at a single pixel i. Consider the following quotation from [1]:

> "Our loss function is inspired by three objectives: prediction should be pixel-wise similar to ground truth maps, therefore a square error loss $||\phi(x_i) - y||^2$ is a reasonable choice.

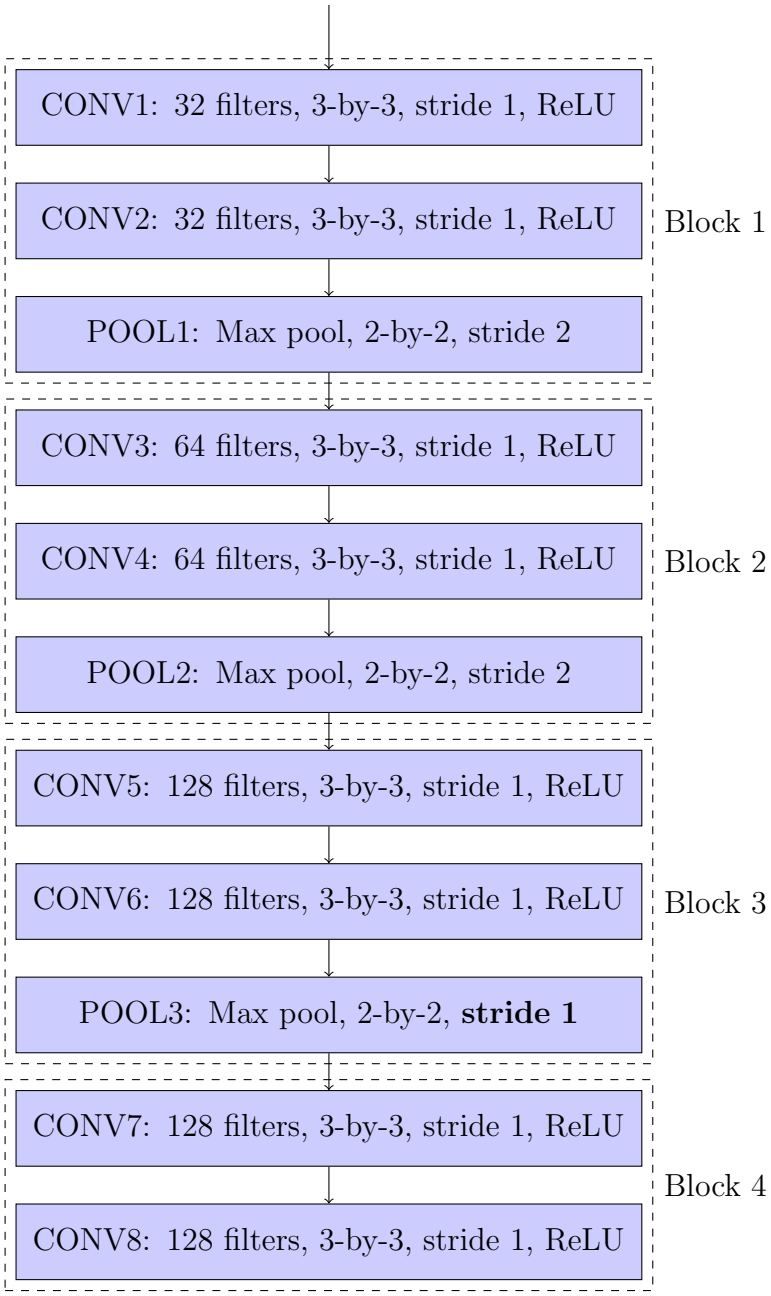

Figure 3: Base CNN.

Secondly, predicted maps should be invariant to their maximum, and there is no point in forcing the network to produce values in a given numerical range, so predictions are normalized by their maximum.

Third, the loss should give the same importance to high and low ground truth values, even though the majority of ground truth pixels are close to zero. For this reason, the deviation between predicted values and ground truth values y_i is weighted by a linear function $\alpha - y_i$, which tends to give more importance to pixels with high ground truth fixation probability."

Due to the above, the paper suggests to use a loss function

$$\frac{1}{N}\sum_{i=1}^{N}\left\|\frac{1}{\alpha - y_i}\cdot\left(\frac{\phi(x_i)}{\max_i \phi(x_i)} - y_i\right)\right\|^2$$

where the sum is over all the N pixels in an image, and where $\alpha = 1.1$.

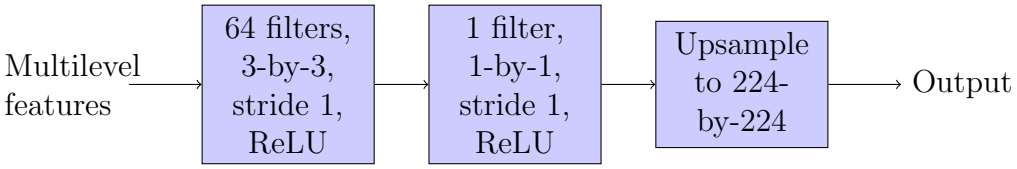

Figure 4: Fixation prediction module.

Implement the loss function above for the network.

- Define the ground truth fixation map as a placeholder of shape $\left[\texttt{None},\ 224,\ 224,\ 1\right]$.
- Find the maxima of the predicted fixation maps for each image in the batch.
  - For example, if you had a batch size of B, the predicted fixation maps' shape would be $\left[B,\ H,\ W,\ 1\right]$. To complete this step you should produce a tensor of shape $\left[B,\ 1,\ 1,\ 1\right]$ where each of the B slices of the tensor has the maximum value of the corresponding predicted fixation map.
  - You will have to use `tf.reduce_max` several times, with the appropriate `keepdims` and `axis` arguments.
- Normalize the activation maps. If you have the dimensions set up correctly for the tensors above, you can write `normalized_outputs = output / output` and TF's broadcasting mechanism will handle taking the division correctly elementwise.
- Calculate the term inside the norm in the equation above.
- Use `tf.square` to calculate the elementwise square of the resulting term, and use `tf.reduce_mean` to calculate the mean (sum and division by N).

1.5 Training

Now that the network and loss function are set up, we can start training the network. This will be the focus of the next exercise, but if you finished all the other tasks you can already start with it.

- Download the fixation prediction dataset for the course project from Moodle, and unzip it to a folder.
- Create the optimizer and training operation in your computational graph.
- Select a batch size B and create a training loop that reads B input images and ground truth fixation maps, and feeds them to the network for training.
- Set up monitoring for the training loss via TensorBoard and terminal printouts.

References

[1] Marcella Cornia, Lorenzo Baraldi, Giuseppe Serra, and Rita Cucchiara. A Deep Multi-Level Network for Saliency Prediction. In *International Conference on Pattern Recognition (ICPR)*, 2016.