Online Appendix for "Neighbouring Prediction for Mortality" ASTIN Bulletin, forthcoming

Online Appendices

A Illustration of Convolution Operation

A.1 Convolution

The convolution operation is a fundamental building block in CNN model. Generally speaking, a convolution operation is a mathematical operation (denoted as $\star$) on two functions (e.g., $x(\cdot)$ and $k(\cdot)$) that produces a new function, $s(\cdot)$, as follows:

$$s(t) \quad = \quad (k \star x)(t) = \int x(t-z)k(z)dz, \tag{A.1}$$

where the function $x(\cdot)$ is the input function; $k(\cdot)$ is called the kernel; and the output function is often referred to as the feature map. For the purpose of application in CNN with 2D input data, a 2D discrete convolution is used:

$$s(t_1, t_2) \quad = \quad (k \star x)(t_1, t_2) = \sum_{z_1, z_2} x(t_1 - z_1, t_2 - z_2)k(z_1, z_2). \tag{A.2}$$

The implementation of the actual convolution operation extracts patches from the input feature map with the kernel, producing an output feature map. This involves sliding the kernel over the input, taking the dot product between the kernel and the respective patch of the input image and finally producing the output, as illustrated in Figure A.1, which shows a convolution operation of 5×5 input feature map and a 3×3 kernel. The upper figure shows the convolution process, and the lower figure shows the detailed calculation results of

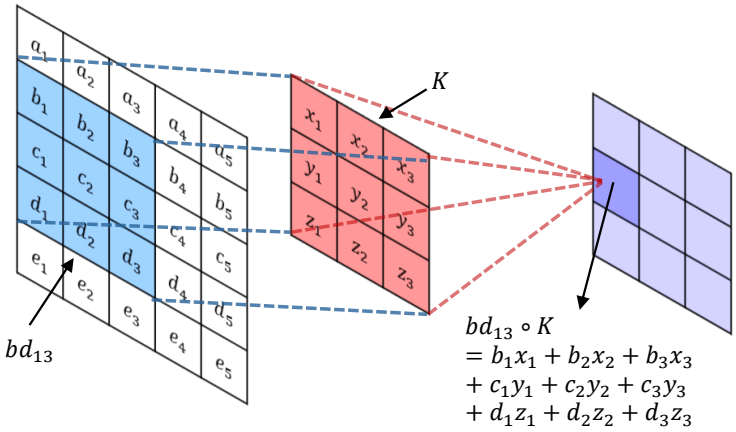
the output figure map. The kernel slides over the input, stopping at each possible location with the same size of the kernel. For example, when the kernel stops at the blue area in the input feature map, using the top-left corner and bottom-right corner to represent the area as bd_{13}, the corresponding output feature map is calculated with the dot product is $bd_{13} \circ K = b_1x_1 + b_2x_2 + b_3x_3 + c_1y_1 + c_2y_2 + c_3y_3 + d_1z_1 + d_2z_2 + d_3z_3$.

A.2 Padding

Note that in the example in Figure A.1, using a a 3×3 kernel, a 5×5 input feature map has a feature map with size 3×3, which is a shrinking output. In general, with a $n \times n$ input feature map and a $k \times k$ kernel, the output feature map has a size of $(n-k+1) \times (n-k+1)$. It's desirable to have an output feature map with the same size so that we do not lose information carried in every layer. *Padding* helps to deal with the shrinking problem. Padding consists of adding an additional border on each side of the input feature map. Zero-padding is the most commonly used padding operation, where zeros are assigned to all the added border, which is illustrated in Figure A.2. We can see that after applying padding, the output feature map of the convolution operator with a 3×3 kernel has the same size of 5 as the original input feature map. In general, the size of the output feature map with a p-column padding, that is, adding p column on each side of the border of the input feature map, is $(n + 2p - k + 1) \times (n + 2p - k + 1)$.

A.3 Stride

Stride provides an other factor to control the output size, and hence, effectively down-sample the input feature maps when needed. Previously we have introduced the convolution operation with the kernel window sliding one tile in each movement so that the centre tile of the convolution windows are contiguous. In fact, the length of each movement is also a parameter to control the convolution, called *stride*. The previous two examples have a stride of 1, i.e., $s = 1$. Strided convolutions are convolution operations with strides higher than 1. Figure A.3 provides a strided convolution example with $s = 2$. The output feature map of the convolution operator with a 3×3 kernel has a size of 2×2. In general, the size of the output feature map with stride s is $(\frac{n+2p-k}{s} + 1) \times (\frac{n+2p-k}{s} + 1)$.



$ac_{13} \circ K$ $= a_1x_1 + a_2x_2$ $+ a_3x_3 + b_1y_1$ $+ b_2y_2 + b_3y_3$ $+ c_1z_1 + c_2z_2$ $+ c_3z_3$	$ac_{24} \circ K$ $= a_2x_1 + a_3x_2$ $+ a_4x_3 + b_2y_1$ $+ b_3y_2 + b_4y_3$ $+ c_2z_1 + c_3z_2$ $+ c_4z_3$	$ac_{35} \circ K$ $= a_3x_1 + a_4x_2$ $+ a_5x_3 + b_3y_1$ $+ b_4y_2 + b_5y_3$ $+ c_3z_1 + c_4z_2$ $+ c_5z_3$
$bd_{13} \circ K$ $= b_1x_1 + b_2x_2$ $+ b_3x_3 + c_1y_1$ $+ c_2y_2 + c_3y_3$ $+ d_1z_1 + d_2z_2$ $+ d_3z_3$	$bd_{24} \circ K$ $= b_2x_1 + b_3x_2$ $+ b_4x_3 + c_2y_1$ $+ c_3y_2 + c_4y_3$ $+ d_2z_1 + d_3z_2$ $+ d_4z_3$	$bd_{35} \circ K$ $= b_3x_1 + b_4x_2$ $+ b_5x_3 + c_3y_1$ $+ c_4y_2 + c_5y_3$ $+ d_3z_1 + d_4z_2$ $+ d_5z_3$
$ce_{13} \circ K$ $= c_1x_1 + c_2x_2$ $+ c_3x_3 + d_1y_1$ $+ d_2y_2 + d_3y_3$ $+ e_1z_1 + e_2z_2$ $+ e_3z_3$	$ce_{24} \circ K$ $= c_2x_1 + c_3x_2$ $+ c_4x_3 + d_2y_1$ $+ d_3y_2 + d_4y_3$ $+ e_2z_1 + e_3z_2$ $+ e_4z_3$	$ce_{35} \circ K$ $= c_3x_1 + c_4x_2$ $+ c_5x_3 + d_3y_1$ $+ d_4y_2 + d_5y_3$ $+ e_3z_1 + e_4z_2$ $+ e_5z_3$

Figure A.1. **An illustration of the convolution operation of 5×5 input feature map and a 3×3 kernel.** The upper figure shows the convolution process, and the lower figure shows the detailed calculation results of the output figure map. The kernel slides over the input, stopping at each possible location with the same size of the kernel. For example, when the kernel stops at the blue area in the input feature map, using the top-left corner and bottom-right corner to represent the area as bd_{13}, the corresponding output feature map is calculated with the dot product is $bd_{13} \circ K = b_1x_1 + b_2x_2 + b_3x_3 + c_1y_1 + c_2y_2 + c_3y_3 + d_1z_1 + d_2z_2 + d_3z_3$.

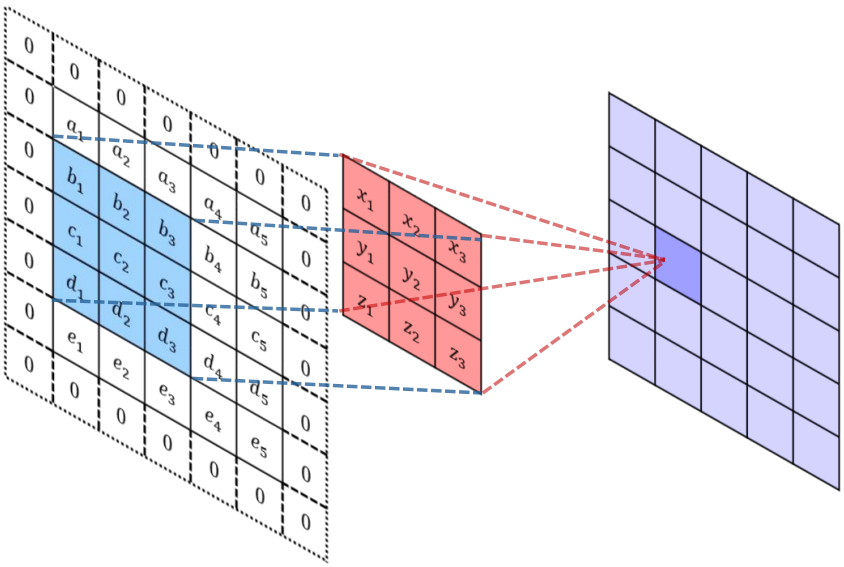


Figure A.2. **An illustration of the convolution operation with padding.** Zero-padding for the convolution operation of 5×5 input feature map and a 3×3 kernel. After applying padding, the output feature map of the convolution operator with a 3×3 kernel has the same size of 5 as the original input feature map.

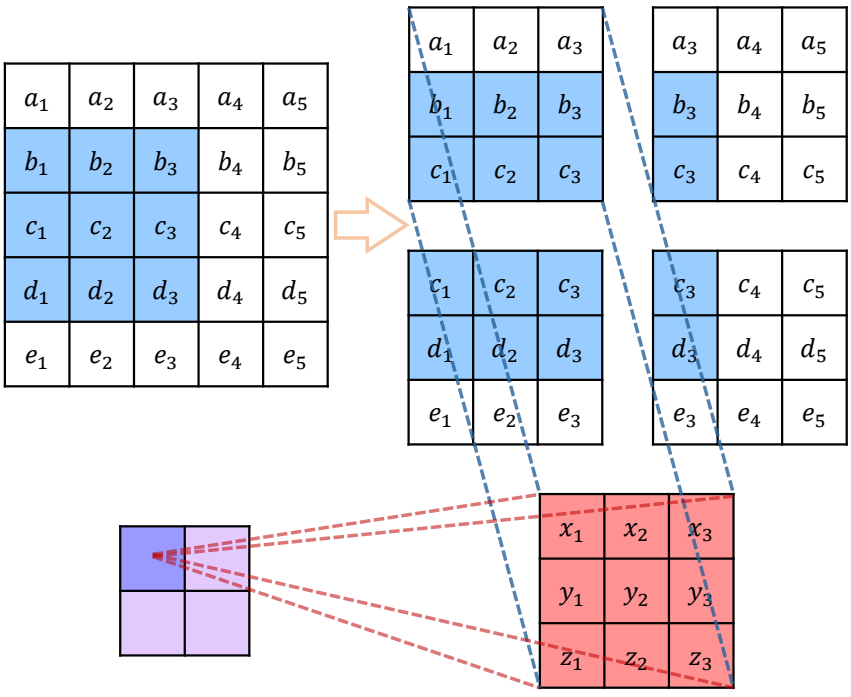


Figure A.3. **An illustration of the convolution stride.** Convolution stride 5×5 input feature map and a 3×3 kernel. The output feature map of the convolution operator with a 3×3 kernel has a size of 2×2.

B Neighbouring Mortality Model Details

Listing 1 and Listing 2 as follows summarize details of CNN structures of Pure Model and Hybrid Model, respectively. In the empirical results of this paper, we set the epochs to be 200 and learning rate to be the default of 0.001. We use cross-validation to tune other tuning parameters. In particular, patience of early stopping is tuned from the interval of $[5, 50]$ with an increment size of 5. We also tried using other complexity constraints such as regulation methods ($\lambda \in (0.00001, 0.0001)$) or dropout method (dropout rate $\in (0, 0.2)$), and we find similar results.

```
# neibourhood mortality images input
images_input <- layer_input(shape = c(x1+x2+1, s, 1), name = 'images')
encoded_images <- layer_conv_2d(images_input, filters = 8,
    kernel_size = c(3,3), padding = "same", activation = 'relu') %>%
    layer_max_pooling_2d(pool_size = c(2,2)) %>% layer_conv_2d(filters = 16,
    kernel_size = c(3,3), padding = "same", activation = 'relu') %>%
    layer_max_pooling_2d(pool_size = c(2,2)) %>% layer_conv_2d(filters = 16,
    kernel_size = c(3,3), padding = "same", activation = 'relu') %>%
    layer_flatten() %>% layer_dense(units = 16, activation = 'relu') %>%
    layer_dense(units = 1)
model <- keras_model(inputs = images_input, outputs = encoded_images)
```

Listing 1. Pure neighbouring mortality model with CNN

```
# neibourhood mortality images input
images_input <- layer_input(shape = c(x1+x2+1, s, 1), name = 'images')
encoded_images <- layer_conv_2d(images_input, filters = 8,
    kernel_size = c(3,3), padding = "same", activation = 'relu') %>%
    layer_max_pooling_2d(pool_size = c(2,2)) %>% layer_conv_2d(filters = 16,
    kernel_size = c(3,3), padding = "same", activation = 'relu') %>%
    layer_max_pooling_2d(pool_size = c(2,2)) %>% layer_conv_2d(filters = 16,
    kernel_size = c(3,3), padding = "same", activation = 'relu') %>%
    layer_flatten() %>% layer_dense(units = 16, activation = 'relu') %>%
    layer_dense(units = 1)

# observable factors features input
features_input <- layer_input(shape = dim(d, name = 'features')
encoded_features <- layer_dense(features_input, units = 4,
    activation = 'relu') %>% layer_dense(units = 1)

# concatenate
concatenated <- layer_concatenate(list(encoded_images, encoded_features))

# output
con_output <- concatenated %>% layer_dense(units = 1)
model <- keras_model(list(images_input, features_input), con_output)
```

Listing 2. Hybrid neighbouring mortality model with CNN