# Chapter 9
# Convolutional Neural Networks


The previous two chapters have been considering fully-connected feed-forward neural (FN) networks and recurrent neural (RN) networks. Fully-connected FN networks are the prototype of networks for deep representation learning on tabular data. This type of networks extracts *global properties* from the features $\boldsymbol{x}$. RN networks are an adaption of FN networks to time-series data. Convolutional neural (CN) networks are a third type of networks, and their specialty is to extract *local structure* from the features. Originally, they have been introduced for speech and image recognition aiming at finding similar structure in different parts of the feature $\boldsymbol{x}$. For instance, if $\boldsymbol{x}$ is a picture consisting of pixels, and if we want to classify this picture according to its contents, then we try to find similar structure (objects) in different locations of this picture. CN networks are suitable for this task as they work with filters (kernels) that have a fixed window size. These filters then screen across the picture to detect similar local structure at different locations in the picture. CN networks were introduced in the 1980s by Fukushima [145] and LeCun et al. [234, 235], and they have been celebrating great success in many applications. Our introduction to CN networks is based on the tutorial of Meier–Wüthrich [269]. For real data applications there are many pre-trained CN network libraries that can be downloaded and used for several different tasks, an example for image recognition is the AlexNet of Krizhevsky et al. [226].

## 9.1 Plain-Vanilla Convolutional Neural Network Layer

Structurally, the CN network architectures are similar to the FN network architectures, only they replace certain FN layers by CN layers. Therefore, we start by introducing the CN layer, and one should keep the structure of the FN layer (7.5)

© The Author(s) 2023

M. V. Wüthrich, M. Merz, *Statistical Foundations of Actuarial Learning and its Applications*, Springer Actuarial, https://doi.org/10.1007/978-3-031-12409-9_9

in mind. In a nutshell, FN layers consider non-linearly activated inner products $\langle \boldsymbol{w}_j^{(m)}, \boldsymbol{z} \rangle$, and CN layers replace these inner products by a type of convolution $\boldsymbol{W}_j^{(m)} * \boldsymbol{z}$.

### 9.1.1 Input Tensors and Channels

We start from an *input tensor* $\boldsymbol{z} \in \mathbb{R}^{q^{(1)} \times \cdots \times q^{(K)}}$ that has dimension $q^{(1)} \times \cdots \times q^{(K)}$. This input tensor $\boldsymbol{z}$ is a *multi-dimensional array of order (length)* $K \in \mathbb{N}$ and with elements $z_{i_1,\ldots,i_K} \in \mathbb{R}$ for $1 \leq i_k \leq q^{(k)}$ and $1 \leq k \leq K$. The special case of order $K = 2$ is a matrix $\boldsymbol{z} \in \mathbb{R}^{q^{(1)} \times q^{(2)}}$. This matrix can illustrate a black and white image of dimension $q^{(1)} \times q^{(2)}$ with the matrix entries $z_{i_1,i_2} \in \mathbb{R}$ describing the intensities of the gray scale in the corresponding pixels $(i_1, i_2)$. A color image typically has the three color channels Red, Green and Blue (RGB), and such a RGB image can be represented by a tensor $\boldsymbol{z} \in \mathbb{R}^{q^{(1)} \times q^{(2)} \times q^{(3)}}$ of order 3 with $q^{(1)} \times q^{(2)}$ being the dimension of the image and $q^{(3)} = 3$ describing the three color channels, i.e., $(z_{i_1,i_2,1}, z_{i_1,i_2,2}, z_{i_1,i_2,3})^\top \in \mathbb{R}^3$ describes the intensities of the colors RGB in the pixel $(i_1, i_2)$.

Typically, the structure of black and white images and RGB images is unified by representing the black and white picture by a tensor $\boldsymbol{z} \in \mathbb{R}^{q^{(1)} \times q^{(2)} \times q^{(3)}}$ of order 3 with a single channel $q^{(3)} = 1$. This philosophy is going to be used throughout this chapter. Namely, if we consider a tensor $\boldsymbol{z} \in \mathbb{R}^{q^{(1)} \times \cdots \times q^{(K-1)} \times q^{(K)}}$ of order $K$, the first $K - 1$ components $(i_1, \ldots, i_{K-1})$ will play the role of the *spatial components* that have a natural topology, and the last components $1 \leq i_K \leq q^{(K)}$ are called the *channels* reflecting, e.g., a gray scale (for $q^{(K)} = 1$) or the RGB intensities (for $q^{(K)} = 3$).

In Sect. 9.1.3, below, we will also study time-series data where we have 2nd order tensors (matrices). The first component reflects time $1 \leq t \leq q^{(1)}$, i.e., the spatial component is temporal for time-series data, and the second component (channels) describes the different elements $\boldsymbol{z}_t = (z_{t,1}, \ldots, z_{t,q^{(2)}})^\top \in \mathbb{R}^{q^{(2)}}$ that are measured/observed at each time point $t$.

### 9.1.2 Generic Convolutional Neural Network Layer

We start from an input tensor $\boldsymbol{z} \in \mathbb{R}^{q_{m-1}^{(1)} \times \cdots \times q_{m-1}^{(K)}}$ of order $K$. The first $K - 1$ components of this tensor have a spatial structure and the $K$-th component stands for the channels. A CN layer applies (local) convolution operations to this tensor. We choose a *filter size*, also called *window size* or *kernel size*, $(f_m^{(1)}, \ldots, f_m^{(K)})^\top \in \mathbb{N}^K$ with $f_m^{(k)} \leq q_{m-1}^{(k)}$, for $1 \leq k \leq K - 1$, and $f_m^{(K)} = q_{m-1}^{(K)}$. This filter size determines

the output dimension of the CN operation by

$$q_m^{(k)} \stackrel{\text{def.}}{=} q_{m-1}^{(k)} - f_m^{(k)} + 1, \tag{9.1}$$

for $1 \le k \le K$. Thus, the size of the image is reduced by the window size of the filter. In particular, the output dimension of the channels component $k = K$ is $q_m^{(K)} = 1$, i.e., all channels are compressed to a scalar output. The spatial components $1 \le k \le K-1$ retain their spatial structure but the dimension is reduced according to (9.1).

A *CN operation* is a mapping (note that the order of the tensor is reduced from $K$ to $K-1$ because the channels are compressed; index $j$ is going to be explained later)

$$\begin{aligned} z_j^{(m)} : \mathbb{R}^{q_{m-1}^{(1)} \times \cdots \times q_{m-1}^{(K)}} &\to \mathbb{R}^{q_m^{(1)} \times \cdots \times q_m^{(K-1)}} \\ z &\mapsto z_j^{(m)}(z) = \left( z^{(m)}_{i_1,\ldots,i_{K-1};j}(z) \right)_{1 \le i_k \le q_m^{(k)}; 1 \le k \le K-1}, \end{aligned} \tag{9.2}$$

taking the values for a fixed activation function $\phi : \mathbb{R} \to \mathbb{R}$

$$z^{(m)}_{i_1,\ldots,i_{K-1};j}(z) = \phi\left( w^{(m)}_{0,j} + \sum_{l_1=1}^{f_m^{(1)}} \cdots \sum_{l_K=1}^{f_m^{(K)}} w^{(m)}_{l_1,\ldots,l_K;j}\, z_{i_1+l_1-1,\ldots,i_{K-1}+l_{K-1}-1,l_K} \right), \tag{9.3}$$

for given intercept $w_{0,j}^{(m)} \in \mathbb{R}$ and *filter weights*

$$\boldsymbol{W}_j^{(m)} = \left( w^{(m)}_{l_1,\ldots,l_K;j} \right)_{1 \le l_k \le f_m^{(k)}; 1 \le k \le K} \in \mathbb{R}^{f_m^{(1)} \times \cdots \times f_m^{(K)}}; \tag{9.4}$$

the network parameter has dimension $r_m = 1 + \prod_{k=1}^{K} f_m^{(k)}$.

At first sight this CN operation looks quite complicated. Let us give some remarks that allow for a better understanding and a more compact notation. The operation in (9.3) chooses the corner $(i_1, \ldots, i_{K-1}, 1)$ as base point, and then it reads the tensor elements in the (discrete) window

$$(i_1, \ldots, i_{K-1}, 1) + \left[0 : f_m^{(1)} - 1\right] \times \cdots \times \left[0 : f_m^{(K-1)} - 1\right] \times \left[0 : f_m^{(K)} - 1\right], \tag{9.5}$$

with given filter weights $\boldsymbol{W}_j^{(m)}$. This window is then moved across the entire tensor $\boldsymbol{z}$ by changing the base point $(i_1, \ldots, i_{K-1}, 1)$ accordingly, but with fixed filter weights $\boldsymbol{W}_j^{(m)}$. This operation resembles a convolution, however, in (9.3) the indices in $z_{i_1+l_1-1,\ldots,i_{K-1}+l_{K-1}-1,l_K}$ run in reverse direction compared to a classical

(mathematical) convolution. By a slight abuse of notation, nevertheless, we use the symbol of the convolution operator $*$ to abbreviate (9.2). This gives us the compact notation:

$$z_j^{(m)} : \mathbb{R}^{q_{m-1}^{(1)} \times \cdots \times q_{m-1}^{(K)}} \to \mathbb{R}^{q_m^{(1)} \times \cdots \times q_m^{(K-1)}}$$
$$z \mapsto z_j^{(m)}(z) = \phi\left(w_{0,j}^{(m)} + \boldsymbol{W}_j^{(m)} * z\right), \tag{9.6}$$

having the activations for $1 \le i_k \le q_m^{(k)}$, $1 \le k \le K-1$,

$$\phi\left(w_{0,j}^{(m)} + \boldsymbol{W}_j^{(m)} * z\right)_{i_1,\ldots,i_{K-1}} = z_{i_1,\ldots,i_{K-1};j}^{(m)}(z),$$

where the latter is given by (9.3).

*Remarks 9.1*

- The beauty of this notation is that we can now see the analogy to the FN layer. Namely, (9.6) exactly plays the role of a FN neuron (7.6), but the CN operation $w_{0,j}^{(m)} + \boldsymbol{W}_j^{(m)} * z$ replaces the inner product $\langle \boldsymbol{w}_j^{(m)}, z\rangle$, and correspondingly accounting for the intercept.
- A FN neuron (7.6) can be seen as a special case of CN operation (9.6). Namely, if we have a tensor of order $K = 1$, the input tensor (vector) reads as $z \in \mathbb{R}^{q_{m-1}^{(1)}}$. That is, we do not have a spatial component, but only $q_{m-1} = q_{m-1}^{(1)}$ channels. In that case we have $\boldsymbol{W}_j^{(m)} * z = \langle \boldsymbol{W}_j^{(m)}, z\rangle$ for the filter weights $\boldsymbol{W}_j^{(m)} \in \mathbb{R}^{q_{m-1}^{(1)}}$, and where we assume that $z$ does not include an intercept component. Thus, the CN operation boils down to a FN neuron in the case of a tensor of order 1.
- In the CN operation we take advantage of having a spatial structure in the tensor $z$, which is not the case in the FN operation. The CN operation takes a spatial input of dimension $\prod_{k=1}^{K} q_{m-1}^{(k)}$ and it maps this input to a spatial object of dimension $\prod_{k=1}^{K-1} q_m^{(k)}$. For this it uses $r_m = 1 + \prod_{k=1}^{K} f_m^{(k)}$ filter weights. The FN operation takes an input of dimension $q_{m-1}$ and it maps it to a 1-dimensional neuron activation, for this it uses $1 + q_{m-1}$ parameters. If we identify the input dimensions $q_{m-1} \stackrel{!}{=} \prod_{k=1}^{K} q_{m-1}^{(k)}$ we can observe that $r_m \ll 1 + q_{m-1}$ because, typically, the filter sizes $f_m^{(k)} \ll q_{m-1}^{(k)}$, for $1 \le k \le K-1$. Thus, the CN operation uses much less parameters as the filters only act locally through the $*$-operation by translating the filter window (9.5).

This understanding now allows us to define a CN layer. Note that the mappings (9.6) have a lower index $j$ which indicates that this is one single projection

(filter extraction), called a *filter*. By choosing multiple different filters $(w_{0,j}^{(m)}, \boldsymbol{W}_j^{(m)})$, we can define the CN layer as follows.

Choose $q_m^{(K)} \in \mathbb{N}$ filters, each having a $r_m$-dimensional filter weight $(w_{0,j}^{(m)}, \boldsymbol{W}_j^{(m)})$, $1 \leq j \leq q_m^{(K)}$. A *CN layer* is a mapping

$$\begin{aligned} \boldsymbol{z}^{(m)} : \mathbb{R}^{q_{m-1}^{(1)} \times \cdots \times q_{m-1}^{(K)}} &\to \mathbb{R}^{q_m^{(1)} \times \cdots \times q_m^{(K)}} \\ \boldsymbol{z} &\mapsto \boldsymbol{z}^{(m)}(\boldsymbol{z}) = \left( z_1^{(m)}(\boldsymbol{z}), \ldots, z_{q_m^{(K)}}^{(m)}(\boldsymbol{z}) \right), \end{aligned} \tag{9.7}$$

with filters $z_j^{(m)}(\boldsymbol{z}) \in \mathbb{R}^{q_m^{(1)} \times \cdots \times q_m^{(K-1)}}$, $1 \leq j \leq q_m^{(K)}$, given by (9.6).

A CN layer (9.7) converts the $q_{m-1}^{(K)}$ input channels to $q_m^{(K)}$ output filters by preserving the spatial structure on the first $K-1$ components of the input tensor $\boldsymbol{z}$. More mathematically, CN layers and networks have been studied, among others, by Zhang et al. [403, 404], Mallat [263] and Wiatowski–Bölcskei [382]. These authors prove that CN networks have certain translation invariance properties and deformation stability. This exactly explains why these networks allow one to recognize similar objects at different locations in the input tensor. Basically, by translating the filter windows (9.5) across the tensor, we try to extract the local structure from the tensor that provides similar signals in different locations of that tensor. Thinking of an image where we try to recognize, say, a dog, such a dog can be located at different sites in the image, and a filter (window) that moves across that image tries to locate the dogs in the image.

A CN layer (9.7) defines one layer indexed by the upper index $^{(m)}$, and for deep representation learning we now have to compose multiple of these CN layers, but we can also compose CN layers with FN layers or RN layers. Before doing so, we need to introduce some special purpose layers and tools that are useful for CN network modeling, this is done in Sect. 9.2, below.

### 9.1.3 Example: Time-Series Analysis and Image Recognition

Most CN network examples are based on time-series data or images. The former has a 1-dimensional temporal component, and the latter has a 2-dimensional spatial component. Thus, these two examples are giving us tensors of orders $K = 2$ and $K = 3$, respectively. We briefly discuss such examples as specific applications of a tensors of a general order $K \geq 2$.

**Time-Series Analysis with CN Networks**

For a time-series analysis we often have observations $\boldsymbol{x}_t \in \mathbb{R}^{q_0}$ for the time points $0 \leq t \leq T$. Bringing this time-series data into a tensor form gives us

$$\boldsymbol{x} \;=\; \boldsymbol{x}_{0:T}^\top \;=\; (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)^\top \;\in\; \mathbb{R}^{(T+1)\times q_0} = \mathbb{R}^{q_0^{(1)} \times q_0^{(2)}},$$

with $q_0^{(1)} = T + 1$ and $q_0^{(2)} = q_0$. We have met such examples in Chap. 8 on RN networks. Thus, for time-series data the input to a CN network is a tensor of order $K = 2$ with a temporal component having the dimension $T + 1$ and at each time point $t$ we have $q_0$ measurements (channels) $\boldsymbol{x}_t \in \mathbb{R}^{q_0}$. A CN network tries to find similar structure at different time points in this time-series data $\boldsymbol{x}_{0:T}$. For a first CN layer $m = 1$ we therefore choose $q_1 \in \mathbb{N}$ filters and consider the mapping

$$\begin{aligned} \boldsymbol{z}^{(1)} : \mathbb{R}^{(T+1)\times q_0} &\to \mathbb{R}^{(T-f_1+2)\times q_1} \\ \boldsymbol{x}_{0:T}^\top &\mapsto \boldsymbol{z}^{(1)}(\boldsymbol{x}_{0:T}^\top) = \left(\boldsymbol{z}_1^{(1)}(\boldsymbol{x}_{0:T}^\top), \ldots, \boldsymbol{z}_{q_1}^{(1)}(\boldsymbol{x}_{0:T}^\top)\right), \end{aligned} \tag{9.8}$$

with filters $\boldsymbol{z}_j^{(1)}(\boldsymbol{x}_{0:T}^\top) \in \mathbb{R}^{T-f_1+2}$, $1 \leq j \leq q_1$, given by (9.6) and for a fixed window size $f_1 \in \mathbb{N}$. From (9.8) we observe that the length of the time-series is reduced from $T + 1$ to $T - f_1 + 2$ accounting for the window size $f_1$. In financial mathematics, a structure (9.8) is often called a rolling window that moves across the time-series $\boldsymbol{x}_{0:T}$ and extracts the corresponding information.

We have introduced two different architectures to process time-series information $\boldsymbol{x}_{0:T}$, and these different architectures serve different purposes. A RN network architecture is most suitable if we try to forecast the next response of a time-series. I.e., we typically process the past observations through a recurrent structure to predict the next response, this is the motivation, e.g., behind Figs. 8.4 and 8.5. The motivation for the use of a CN network architecture is different as we try to find similar structure at different times, e.g., in a financial time-series we may be interested in finding the downturns of more than 20%. The latter is a local analysis which is explored by local filters (of a finite window size).

**Image Recognition**

Image recognition extends (9.8) by one order to a tensor of order $K = 3$. Typically, we have images of dimensions (pixels) $I \times J$, and having three color channels RGB. These images then read as

$$\boldsymbol{x} \;=\; (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) \;\in\; \mathbb{R}^{I\times J\times 3} = \mathbb{R}^{q_0^{(1)} \times q_0^{(2)} \times q_0^{(3)}},$$

where $\boldsymbol{x}_1 \in \mathbb{R}^{I\times J}$ is the intensity of red, $\boldsymbol{x}_2 \in \mathbb{R}^{I\times J}$ is the intensity of green, and $\boldsymbol{x}_3 \in \mathbb{R}^{I\times J}$ is the intensity of blue.

Chose a window size of $f_1^{(1)} \times f_1^{(2)}$ and $q_1 \in \mathbb{N}$ filters to receive the CN layer

$$\begin{aligned} \boldsymbol{z}^{(1)} &: \mathbb{R}^{I\times J\times 3} \to \mathbb{R}^{(I-f_1^{(1)}+1)\times(J-f_1^{(2)}+1)\times q_1} \\ (\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3) &\mapsto \boldsymbol{z}^{(1)}(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3) = \left(\boldsymbol{z}_1^{(1)}(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3),\ldots,\boldsymbol{z}_{q_1}^{(1)}(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3)\right), \end{aligned} \tag{9.9}$$

with filters $\boldsymbol{z}_j^{(1)}(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3) \in \mathbb{R}^{(I-f_1^{(1)}+1)\times(J-f_1^{(2)}+1)}$, $1 \le j \le q_1$. Thus, we compress the 3 channels in each filter $j$, but we preserve the spatial structure of the image (by the convolution operation $*$).

For black and white pictures which only have one color channel, we preserve the spatial structure of the picture, and we modify the input tensor to a tensor of order 3 and of the form

$$\boldsymbol{x} = (\boldsymbol{x}_1) \in \mathbb{R}^{I\times J\times 1}.$$

## 9.2 Special Purpose Tools for Convolutional Neural Networks

### *9.2.1 Padding with Zeros*

We have seen that the CN operation reduces the size of the output by the filter sizes, see (9.1). Thus, if we start from an image of size $100 \times 50 \times 1$, and if the filter sizes are given by $f_m^{(1)} = f_m^{(2)} = 9$, then the output will be of dimension $92 \times 42 \times q_1^{(3)}$, see (9.9). Sometimes, this reduction in dimension is impractical, and padding helps to keep the original shape. Padding a tensor $\boldsymbol{z}$ with $p_m^{(k)}$ parameters, $1 \le k \le K-1$, means that the tensor is extended in all $K-1$ spatial directions by (typically) adding zeros of that size, so that the padded tensor has dimension

$$\left(p_m^{(1)} + q_{m-1}^{(1)} + p_m^{(1)}\right) \times \cdots \times \left(p_m^{(K-1)} + q_{m-1}^{(K-1)} + p_m^{(K-1)}\right) \times q_{m-1}^{(K)}.$$

This implies that the output filters will have the dimensions

$$q_m^{(k)} = q_{m-1}^{(k)} + 2p_m^{(k)} - f_m^{(k)} + 1,$$

for $1 \le k \le K-1$. The spatial dimension of the original tensor size is preserved if $2p_m^{(k)} - f_m^{(k)} + 1 = 0$. Padding does not add any additional parameters, but it is only used to reshape the tensors.

### 9.2.2 Stride

Strides are used to skip part of the input tensor $z$ in order to reduce the size of the output. This may be useful if the input tensor is a very high resolution image. Choose the stride parameters $s_m^{(k)}$, $1 \leq k \leq K-1$. We can then replace the summation in (9.3) by the following term

$$\sum_{l_1=1}^{f_m^{(1)}} \cdots \sum_{l_K=1}^{f_m^{(K)}} w_{l_1,\ldots,l_K;j}^{(m)} \, z_{s_m^{(1)}(i_1-1)+l_1,\ldots,s_m^{(K-1)}(i_{K-1}-1)+l_{K-1},l_K}.$$

This only extracts the tensor entries on a discrete grid of the tensor by translating the window by multiples of integers, see also (9.5),

$$\left(s_m^{(1)}(i_1-1),\ldots,s_m^{(K-1)}(i_{K-1}-1),1\right)+\left[1:f_m^{(1)}\right]\times\cdots\times\left[1:f_m^{(K-1)}\right]\times\left[0:f_m^{(K)}-1\right],$$

and the size of the output is reduced correspondingly. If we choose strides $s_m^{(k)} = f_m^{(k)}$, $1 \leq k \leq K-1$, we receive a partition of the spatial part of the input tensor $z$, this is going to be used in the max-pooling layer (9.11).

### 9.2.3 Dilation

Dilation is similar to stride, though, different in that it enlarges the filter sizes instead of skipping certain positions in the input tensor. Choose the dilation parameters $e_m^{(k)}$, $1 \leq k \leq K-1$. We can then replace the summation in (9.3) by the following term

$$\sum_{l_1=1}^{f_m^{(1)}} \cdots \sum_{l_K=1}^{f_m^{(K)}} w_{l_1,\ldots,l_K;j}^{(m)} \, z_{i_1+e_m^{(1)}(l_1-1),\ldots,i_{K-1}+e_m^{(K-1)}(l_{K-1}-1),l_K}.$$

This applies the filter weights to the tensor entries on discrete grids

$$(i_1,\ldots,i_{K-1},1)+e_m^{(1)}\left[0:f_m^{(1)}-1\right]\times\cdots\times e_m^{(K-1)}\left[0:f_m^{(K-1)}-1\right]\times\left[0:f_m^{(K)}-1\right],$$

where the intervals $e_m^{(k)}[0:f_m^{(k)}-1]$ run over the grids of span sizes $e_m^{(k)}$, $1 \leq k \leq K-1$. Thus, in comparably smoothing images we do not read all the pixels but only every $e_m^{(k)}$-th pixel in the window. Also this reduces the size of the output tensor.

### 9.2.4 Pooling Layer

As we have seen above, the dimension of the tensor is reduced by the filter size in each spatial direction if we do not apply padding with zeros. In general, deep representation learning follows the paradigm of auto-encoding by reducing a high-dimensional input to a low-dimensional representation. In CN networks this is usually (efficiently) done by so-called pooling layers. In spirit, pooling layers work similarly to CN layers (having a fixed window size), but we do not apply a convolution operation $*$, but rather a maximum operation to the window to extract the dominant tensor elements.

We choose a fixed window size $(f_m^{(1)}, \ldots, f_m^{(K-1)})^\top \in \mathbb{N}^{K-1}$ and strides $s_m^{(k)} = f_m^{(k)}$, $1 \le k \le K-1$, for the spatial components of the tensor $z$ of order $K$. A *max-pooling layer* is given by

$$\begin{aligned} z^{(m)} : \mathbb{R}^{q_{m-1}^{(1)} \times \cdots \times q_{m-1}^{(K)}} &\to \mathbb{R}^{q_m^{(1)} \times \cdots \times q_m^{(K)}} \\ z &\mapsto z^{(m)}(z) = \mathrm{MaxPool}(z), \end{aligned} \tag{9.10}$$

with dimensions $q_m^{(K)} = q_{m-1}^{(K)}$ and for $1 \le k \le K-1$

$$q_m^{(k)} = \left\lfloor q_{m-1}^{(k)} / f_m^{(k)} \right\rfloor, \tag{9.11}$$

having the activations for $1 \le i_k \le q_m^{(k)}$, $1 \le k \le K$,

$$\mathrm{MaxPool}(z)_{i_1,\ldots,i_K} = \max_{\substack{1 \le l_k \le f_m^{(k)}, \\ 1 \le k \le K-1}} z_{f_m^{(1)}(i_1-1)+l_1, \ldots, f_m^{(K-1)}(i_{K-1}-1)+l_{K-1}, i_K}.$$

Alternatively, the floors in (9.11) could be replaced by ceilings and padding with zeros to receive the right cardinality. This extracts the maximums from the (spatial) windows

$$\begin{aligned} &\left(f_m^{(1)}(i_1-1), \ldots, f_m^{(K-1)}(i_{K-1}-1), i_K\right) + \left[1 : f_m^{(1)}\right] \times \cdots \times \left[1 : f_m^{(K-1)}\right] \times [0] \\ &= \left[f_m^{(1)}(i_1-1)+1 : f_m^{(1)} i_1\right] \times \cdots \times \left[f_m^{(K-1)}(i_{K-1}-1)+1 : f_m^{(K-1)} i_{K-1}\right] \times [i_K], \end{aligned}$$

for each channel $1 \le i_K \le q_{m-1}^{(K)}$ individually. Thus, the max-pooling operator is chosen such that it extracts the maximum of each channel and each window, the windows providing a partition of the spatial part of the tensor. This reduces the dimension of the tensor according to (9.11), e.g., if we consider a tensor of order 3 of an RGB image of dimension $I \times J = 180 \times 50$ and apply a max-pooling layer with window sizes $f_m^{(1)} = 10$ and $f_m^{(2)} = 5$, we receive a dimension reduction

$$180 \times 50 \times 3 \;\mapsto\; 18 \times 10 \times 3.$$

Replacing the maximum operator in (9.10) by an averaging operator is sometimes also used, and this is called an *average-pooling layer*.

### 9.2.5 Flatten Layer

A *flatten layer* performs the transformation of rearranging a tensor to a vector, so that the output of a flatten layer can be used as an input to a FN layer. That is,

$$\begin{aligned} z^{(m)} : \mathbb{R}^{q_{m-1}^{(1)} \times \cdots \times q_{m-1}^{(K)}} &\to \mathbb{R}^{q_m} \\ z &\mapsto z^{(m)}(z) = \left( z_{1,\ldots,1}, \ldots, z_{q_{m-1}^{(1)},\ldots,q_{m-1}^{(K)}} \right)^\top, \end{aligned} \tag{9.12}$$

with $q_m = \prod_{k=1}^{K} q_{m-1}^{(k)}$. We have already used flatten layers after embedding layers on lines 8 and 11 of Listing 7.4.

## 9.3 Convolutional Neural Network Architectures

### 9.3.1 Illustrative Example of a CN Network Architecture

We are now ready to patch everything together. Assume we have RGB images described by tensors $\boldsymbol{x}^{(0)} \in \mathbb{R}^{I \times J \times 3}$ of order 3 modeling the three RGB channels of images of a fixed size $I \times J$. Moreover, we have the tabular feature information $\boldsymbol{x}^{(1)} \in \mathcal{X} \subset \{1\} \times \mathbb{R}^q$ that describes further properties of the data. That is, we have an input variable $(\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)})$, and we aim at predicting a response variable $Y$ by a using a suitable regression function

$$(\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}) \;\mapsto\; \mu(\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}) = \mathbb{E}\left[ Y \,\middle|\, \boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)} \right]. \tag{9.13}$$

We choose two convolutional layers $z^{(\mathrm{CN1})}$ and $z^{(\mathrm{CN2})}$, each followed by a max-pooling layer $z^{(\mathrm{Max1})}$ and $z^{(\mathrm{Max2})}$, respectively. Then we apply a flatten layer $z^{(\mathrm{flatten})}$ to bring the learned representation into a vector form. These layers are chosen according to (9.7), (9.10) and (9.12) with matching input and output dimensions so that the following composition is well-defined

$$z^{(5:1)} = \left( z^{(\mathrm{flatten})} \circ z^{(\mathrm{Max2})} \circ z^{(\mathrm{CN2})} \circ z^{(\mathrm{Max1})} \circ z^{(\mathrm{CN1})} \right) : \mathbb{R}^{I \times J \times 3} \to \mathbb{R}^{q_5}.$$

Listing 9.1 provides an example starting from a $I \times J \times 3 = 180 \times 50 \times 3$ input tensor $\boldsymbol{x}^{(0)}$ and receiving a $q_5 = 60$ dimensional learned representation $z^{(5:1)}(\boldsymbol{x}^{(0)}) \in \mathbb{R}^{60}$.

**Listing 9.1** CN network architecture in `keras`

```
shape <- c(180,50,3)
#
model = keras_model_sequential()
model %>%
  layer_conv_2d(filters = 10, kernel_size = c(11,6), activation='tanh',
                                                     input_shape = shape) %>%
  layer_max_pooling_2d(pool_size = c(10,5)) %>%
  layer_conv_2d(filters = 5, kernel_size = c(6,4), activation='tanh') %>%
  layer_max_pooling_2d(pool_size = c(3,2)) %>%
  layer_flatten()
```

**Listing 9.2** Summary of CN network architecture

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 170, 45, 10)       1990
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 17, 9, 10)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 6, 5)          1205
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 4, 3, 5)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 60)                0
=================================================================
Total params: 3,195
Trainable params: 3,195
Non-trainable params: 0
```

Listing 9.2 gives the summary of this architecture providing the dimension reduction mappings (encodings)

$$180 \times 50 \times 3 \overset{\text{CN1}}{\mapsto} 170 \times 45 \times 10 \overset{\text{Max1}}{\mapsto} 17 \times 9 \times 10 \overset{\text{CN2}}{\mapsto} 12 \times 6 \times 5 \overset{\text{Max2}}{\mapsto} 4 \times 3 \times 5 \overset{\text{flatten}}{\mapsto} 60.$$

The first CN layer ($m = 1$) involves $q_1^{(3)} r_1 = 10 \cdot (1 + 11 \cdot 6 \cdot 3) = 1'990$ filter weights $(w_{0,j}^{(1)}, \boldsymbol{W}_j^{(1)})_{1 \le j \le q_1^{(3)}}$ (including the intercepts), and the second CN layer ($m = 3$) involves $q_3^{(3)} r_3 = 5 \cdot (1 + 6 \cdot 4 \cdot 10) = 1'205$ filter weights $(w_{0,j}^{(3)}, \boldsymbol{W}_j^{(3)})_{1 \le j \le q_3^{(3)}}$. Altogether we have a network parameter of dimension $3'195$ to be fitted in this CN network architecture.

To perform the prediction task (9.13) we concatenate the learned representation $\boldsymbol{z}^{(5:1)}(\boldsymbol{x}^{(0)}) \in \mathbb{R}^{q_5}$ of the RGB image $\boldsymbol{x}^{(0)}$ with the tabular feature $\boldsymbol{x}^{(1)} \in \mathcal{X} \subset \{1\} \times \mathbb{R}^q$. This concatenated vector is processed through a FN network architecture $\boldsymbol{z}^{(d+5:6)}$ of depth $d \ge 1$ providing the output

$$\left(\boldsymbol{z}^{(5:1)}(\boldsymbol{x}^{(0)}), \boldsymbol{x}^{(1)}\right) \;\mapsto\; \mathbb{E}\left[Y \,\middle|\, \boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}\right] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d+5:6)}\left(\boldsymbol{z}^{(5:1)}(\boldsymbol{x}^{(0)}), \boldsymbol{x}^{(1)}\right)\right\rangle,$$

for given link function $g$. This last step can be done in complete analogy to Chap. 7, and fitting of such a network architecture uses variants of the SGD algorithm.

### 9.3.2 Lab: Telematics Data

We present a CN network example that studies time-series of telematics car driving data. Unfortunately, this data is not publicly available. Recently, telematics car driving data has gained much popularity in actuarial science, because this data provides information of car drivers that goes beyond the classical features (age of driver, year of driving test, etc.), and it provides a better discrimination of good and bad drivers as it is directly based on the driving habits and the driving styles.

The telematics data has many different aspects. Raw telematics data typically consists of high-frequency GPS location data, say, second by second, from which several different statistics such as speed, acceleration and change of direction can be calculated. Besides the GPS location data, it often contains vehicle speeds from the vehicle instrumental panel, and acceleration in all directions from an accelerometer. Thus, often, there are 3 different sources from which the speed and the acceleration can be extracted. In practice, the data quality is often an issue as these 3 different sources may give substantially different numbers, Meng et al. [271] give a broader discussion on these data quality issues. The telematics GPS data is often complemented by further information such as engine revolutions, daytime of trips, road and traffic conditions, weather conditions, traffic rule violations, etc. This raw telematics data is then pre-processed, e.g., special maneuvers are extracted (speeding, sudden acceleration, hard braking, extreme right- and left-turns), total distances are calculated, driving distances at different daytimes and weekdays are analyzed. For references analyzing such statistics for predictive modeling we refer to Ayuso et al. [17–19], Boucher et al. [42], Huang–Meng [193], Lemaire et al. [246], Paefgen et al. [291], So et al. [344], Sun et al. [347] and Verbelen et al. [370]. A different approach has been taken by Wüthrich [388] and Gao et al. [151, 154, 155], namely, these authors aggregate the telematics data of speed and acceleration to so-called speed-acceleration $v$-$a$ heatmaps. These $v$-$a$ heatmaps are understood as images which can be analyzed, e.g., by CN networks; such an analysis has been performed in Zhu–Wüthrich [407] for image classification and in Gao et al. [154] for claim frequency modeling. Finally, the work of Weidner et al. [377, 378] directly acts on the time-series of the telematics GPS data by performing a Fourier analysis.

In this section, we aim at allocating individual car driving trips to the right drivers by directly analyzing the time-series of the telematics data of these trips using CN networks. We therefore replicate the analysis of Gao–Wüthrich [156] on slightly different data. For our illustrative example we select 3 car drivers and we call them driver A, driver B and driver C. For each of these 3 drivers we choose individual car driving trips of 180 seconds, and we analyze their speed-acceleration-change in angle ($v$-$a$-$\Delta$) pattern every second. Thus, for $t = 1, \ldots, T = 180$, we study the three input channels

$$\boldsymbol{x}_{s,t} = (v_{s,t}, a_{s,t}, \Delta_{s,t})^\top \in [2, 50]\text{km/h} \times [-3, 3]\text{m/s}^2 \times [0, 1/2] \subset \mathbb{R}^3,$$

where $1 \leq s \leq S$ labels all individual trips of the considered drivers. This data has been pre-processed by cutting-out the idling phase and the speeds above 50km/h and concatenating the remaining pieces. We perform this pre-processing since we do not want to identify the drivers because they have a special idling phase picture or because they are more likely on the highway. Acceleration has been censored at $\pm 3\text{m/s}^2$ because we cannot exclude that more extreme observations are caused by data quality issues (note that the acceleration is calculated from the GPS coordinates and if the signals are not fully precise it can lead to extreme acceleration observations). Finally, change in angle is measured in absolute values of sine per second (censored at 1/2), i.e., we do not distinguish between left and right turns. This then provides us with three time-series channels giving tensors of order 2

$$\boldsymbol{x}_s = \left( (v_{s,1}, a_{s,1}, \Delta_{s,1})^\top, \ldots, (v_{s,180}, a_{s,180}, \Delta_{s,180})^\top \right)^\top \in \mathbb{R}^{180 \times 3},$$

for $1 \leq s \leq S$. Moreover, there is a categorical response $Y_s \in \{\text{A}, \text{B}, \text{C}\}$ indicating which driver has been driving trip $s$.

Figure 9.1 illustrates the first three trips $\boldsymbol{x}_s$ of $T = 180$ seconds of each of these three drivers A (top), B (middle) and C (bottom); note that the 180 seconds have been chosen at a random location within each trip. The first lines in red color show the acceleration patterns $(a_t)_{1 \leq t \leq T}$, the second lines in black color the change in angle patterns $(\Delta_t)_{1 \leq t \leq T}$, and the last lines in blue color the speed patterns $(v_t)_{1 \leq t \leq T}$.

Table 9.1 summarizes the available data. In total we have 932 individual trips, and we randomly split these trips in the learning data $\mathcal{L}$ consisting of 744 trips and the test data $\mathcal{T}$ collecting the remaining trips. The goal is to train a classification model that correctly allocates the test data $\mathcal{T}$ to the right driver. As feature information, we use the telematics data $\boldsymbol{x}_s$ of length 180 seconds. We design a logistic categorical regression model with response set $\mathcal{Y} = \{\text{A}, \text{B}, \text{C}\}$. Hence, we obtain a vector-valued parameter EF with a response having 3 levels, see Sect. 2.1.4.

To process the telematics data $\boldsymbol{x}_s$, we design a CN network architecture having three convolutional layers $z^{(\text{CN}j)}$, $1 \leq j \leq 3$, each followed by a max-pooling layer $z^{(\text{Max}j)}$, then we apply a drop-out layer $z^{(\text{DO})}$ and finally a fully-connected FN layer $z^{(\text{FN})}$ providing the logistic response classification; this is the same network architecture as used in Gao–Wüthrich [156]. The code is given in Listing 9.3 and it describes the mapping

$$z^{(8:1)} = \left( z^{(\text{FN})} \circ z^{(\text{DO})} \circ z^{(\text{Max3})} \circ z^{(\text{CN3})} \circ z^{(\text{Max2})} \circ z^{(\text{CN2})} \circ z^{(\text{Max1})} \circ z^{(\text{CN1})} \right) : \mathbb{R}^{T \times 3} \to (0,1)^3.$$

The first CN and pooling layer $z^{(\text{Max1})} \circ z^{(\text{CN1})}$ maps the dimension $180 \times 3$ to a tensor of dimension $58 \times 12$ using 12 filters; the max-pooling uses the floor (9.11). The second CN and pooling layer $z^{(\text{Max2})} \circ z^{(\text{CN2})}$ maps to $18 \times 10$ using 10 filters, and the third CN and pooling layer $z^{(\text{Max3})} \circ z^{(\text{CN3})}$ maps to $1 \times 8$ using 8 filters. Actually, this last max-pooling layer is a global max-pooling layer extracting the maximum in each of the 8 filters. Next, we apply a drop-out layer with a drop-out

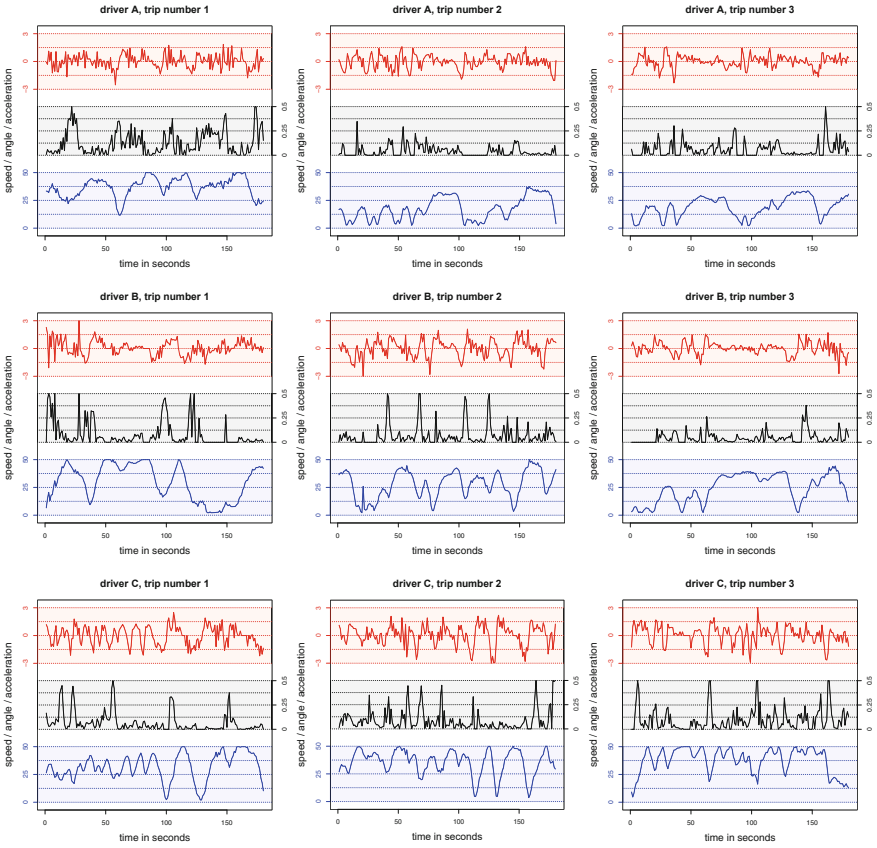


**Fig. 9.1** First 3 trips of driver A (top), driver B (middle) and driver C (bottom); each trip is 180 seconds, red color shows the acceleration pattern $(a_t)_t$, black color the change in angle pattern $(\Delta_t)_t$ and blue color the speed pattern $(v_t)_t$

**Table 9.1** Summary of the trips and the choice of learning and test data sets $\mathcal{L}$ and $\mathcal{T}$

| | Driver A | Driver B | Driver C | Total |
|---|---|---|---|---|
| Number of trips $S$ | 261 | 385 | 286 | 932 |
| Learning data $\mathcal{L}$ | 209 | 307 | 228 | 744 |
| Test data $\mathcal{T}$ | 52 | 78 | 58 | 188 |
| Average speed $v_t$ | 24.8 | 30.4 | 30.2 | km/h |
| Average acceleration/braking $\lvert a_t \rvert$ | 0.56 | 0.61 | 0.74 | $m/s^2$ |
| Average change in angle $\Delta_t$ | 0.065 | 0.054 | 0.076 | $\lvert \sin \rvert$/s |

rate of 30% to prevent from over-fitting. Finally we apply a fully-connected FN layer that maps the 8 neurons to the 3 categorical outputs using the softmax output activation function, which provides the canonical link of the logistic categorical EF.

**Listing 9.3** CN network architecture for the individual car trip allocation

```
shape <- c(180,3)
#
model = keras_model_sequential()
model %>%
layer_conv_1d(filters = 12, kernel_size = 5, activation='tanh',
                                                    input_shape = shape) %>%
layer_max_pooling_1d(pool_size = 3) %>%
layer_conv_1d(filters = 10, kernel_size = 5, activation='tanh') %>%
layer_max_pooling_1d(pool_size = 3) %>%
layer_conv_1d(filters = 8, kernel_size = 5, activation='tanh') %>%
layer_global_max_pooling_1d() %>%
layer_dropout(rate = .3) %>%
layer_dense(units = 3, activation = 'softmax')
```

For a summary of the network architecture see Listing 9.4. Altogether this involves 1'237 network parameters that need to be fitted.

**Listing 9.4** Summary of CN network architecture for the individual car trip allocation

```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 176, 12)           192
_________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 58, 12)         0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 54, 10)            610
_________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 18, 10)         0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 14, 8)             408
_________________________________________________________________
global_max_pooling1d_1 (GlobalMaxPool (None, 8)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 27
=================================================================
Total params: 1,237
Trainable params: 1,237
Non-trainable params: 0
```

We choose the 744 trips of the learning data $\mathcal{L}$ to train this network to the classification task, see Table 9.1. We use the multi-class cross-entropy loss function, see (4.19), with 80% of the learning data $\mathcal{L}$ as training data $\mathcal{U}$ and the remaining 20% as validation data $\mathcal{V}$ to track over-fitting. We retrieve the network with the smallest validation loss using a callback, we refer to Listing 7.3 for a callback. Since the learning data is comparably small and to reduce randomness, we use the nagging predictor averaging over 10 different network fits (using different seeds).

**Table 9.2** Out-of-sample confusion matrix

| | True labels | | |
|---|---|---|---|
| | Driver A | Driver B | Driver C |
| Predicted label A | **39** | 10 | 2 |
| Predicted label B | 9 | **66** | 6 |
| Predicted label C | 4 | 2 | **50** |
| % correctly allocated | 75.0% | 84.6% | 86.2% |
| # of trips in test data | 52 | 78 | 58 |

These fitted networks then provide us with a mapping

$$z^{(8:1)} : \mathbb{R}^{T\times 3} \to (0,1)^3, \qquad \boldsymbol{x} \mapsto z^{(8:1)}(\boldsymbol{x}) = \left(z_{\mathrm{A}}^{(8:1)}(\boldsymbol{x}), z_{\mathrm{B}}^{(8:1)}(\boldsymbol{x}), z_{\mathrm{C}}^{(8:1)}(\boldsymbol{x})\right)^\top,$$

and for each trip $\boldsymbol{x}_s \in \mathbb{R}^{T\times 3}$ we receive the classification

$$\widehat{Y}_s \;=\; \underset{y\in\{\mathrm{A},\mathrm{B},\mathrm{C}\}}{\arg\max}\; z_y^{(8:1)}(\boldsymbol{x}_s).$$

Table 9.2 shows the out-of-sample results on the test data $\mathcal{T}$. On average more than 80% of all trips are correctly allocated; a purely random allocation would provide a success rate of 33%. This shows that this allocation problem can be solved rather successfully and, indeed, the CN network architecture is able to learn structure in the telematics trip data $\boldsymbol{x}_s$ that allows one to discriminate car drivers. This sounds very promising. In fact, the telematics car driving data seems to be very transparent which, of course, also raises privacy issues. On the downside we should mention that from this approach we cannot really see what the network has learned and how it manages to distinguish the different trips.

There are several approaches that try to visualize what the network has learned in the different layers by extracting the filter activations in the CN layers, others try to invert the networks trying to backtrack which activations and weights mostly contribute to a certain output, we mention, e.g., DeepLIFT of Shrikumar et al. [339]. For more analysis and references we refer to Sect. 4 of the tutorial Meier–Wüthrich [269]. We do not further discuss this and close this example.

### *9.3.3 Lab: Mortality Surface Modeling*

We revisit the mortality example of Sect. 8.4.2 where we used a LSTM architecture to process the raw mortality data for forecasting, see Fig. 8.13. We are going to do a (small) change to that architecture by simply replacing the LSTM encoder by a CN network encoder. This approach has been promoted in the literature, e.g., by Perla et al. [301], Schnürch–Korn [330] and Wang et al. [375]. A main difference between these references is whether the mortality tensor is considered as a tensor

of order 2 (reflecting time-series data) or of order 3 (reflecting the mortality surface as an image). In the present example we are going to interpret the mortality tensor as a monochrome image, and this requires that we extend (8.23) by an additional channels component

$$\begin{aligned}\boldsymbol{x}_{t-\tau:t-1} &= (\boldsymbol{x}_{t-\tau}, \ldots, \boldsymbol{x}_{t-1})^\top \\ &= \left(M_{x,s}\right)_{t-\tau \le s \le t-1, x_0 \le x \le x_1} \in \mathbb{R}^{\tau \times (x_1 - x_0 + 1) \times 1} = \mathbb{R}^{5 \times 100 \times 1},\end{aligned}$$

for a lookback period of $\tau = 5$. The LSTM cell encodes this tensor/matrix into a 20-dimensional vector which is then concatenated with the embeddings of the country code and the gender code (8.24). We use the same architecture here, only the LSTM part is replaced by a CN network in (8.25), the corresponding code is given on lines 14–17 of Listing 9.5.

**Listing 9.5** CN network architecture to directly process the raw mortality rates $(M_{x,t})_{x,t}$

```
1  Tensor  = layer_input(shape=c(lookback,100,1), dtype='float32', name='Tensor')
2  Country = layer_input(shape=c(1), dtype='int32',   name='Country')
3  Gender  = layer_input(shape=c(1), dtype='int32',   name='Gender')
4  Time    = layer_input(shape=c(1), dtype='float32', name='Time')
5  #
6  CountryEmb = Country %>%
7   layer_embedding(input_dim=8,output_dim=1,input_length=1,name='CountryEmb') %>%
8   layer_flatten(name='Country_flat')
9  #
10 GenderEmb = Gender %>%
11  layer_embedding(input_dim=2,output_dim=1,input_length=1,name='GenderEmb') %>%
12  layer_flatten(name='Gender_flat')
13 #
14 CN = Tensor %>%
15  layer_conv_2d(filter = 10, kernel_size = c(5,5), activation = 'linear') %>%
16                  layer_max_pooling_2d(pool_size = c(1,8)) %>%
17                  layer_flatten()
18 #
19 Output = list(CN,CountryEmb,GenderEmb) %>% layer_concatenate() %>%
20  layer_dense(units=100, activation='linear',  name='scalarproduct') %>%
21  layer_reshape(c(1,100), name = 'Output')
22 #
23 model = keras_model(inputs = list(Tensor, Country, Gender),
24                     outputs = c(Output))
```

Line 15 maps the input tensor $5 \times 100 \times 1$ to a tensor $1 \times 96 \times 10$ having 10 filters, the max-pooling layer reduces this tensor to $1 \times 12 \times 10$, and the flatten layer encodes this tensor into a 120-dimensional vector. This vector is then concatenated with the embedding vectors of the country and the gender codes, and this provides us with $r = 12'570$ network parameters, thus, the LSTM architecture and the CN network architecture use roughly equally many network parameters that need to be fitted. We then use the identical partition in training, validation and test data as in Sect. 8.4.2, i.e., we use the data from 1950 to 2003 for fitting the network architecture, which is then used to forecast the calendar years 2004 to 2018. The results are presented in Table 9.3.

**Table 9.3** Comparison of the out-of-sample mean squared losses for the calendar years $2004 \leq t \leq 2018$; the figures are in $10^{-4}$

| | Female | | | Male | | |
|---|---|---|---|---|---|---|
| | LC | LSTM | CN | LC | LSTM | CN |
| Austria AUT | 0.765 | **0.312** | 0.635 | 2.527 | **1.169** | 1.569 |
| Belgium BE | 0.371 | 0.311 | **0.290** | 2.835 | **0.960** | 1.100 |
| Switzerland CH | 0.654 | **0.478** | 0.772 | 1.609 | **1.134** | 2.035 |
| Spain ESP | 1.446 | 0.514 | **0.199** | 1.742 | 0.245 | **0.240** |
| France FRA | **0.175** | 1.684 | 0.309 | **0.333** | 0.363 | 0.770 |
| Italy ITA | **0.179** | 0.330 | 0.186 | 0.874 | **0.320** | 0.421 |
| The Netherlands NL | 0.426 | 0.315 | **0.266** | 1.978 | **0.601** | 0.606 |
| Portugal POR | 2.097 | 0.464 | **0.416** | 1.848 | **1.239** | 1.880 |

We observe that in our case the CN network architecture provides good results for the female populations, whereas for the male populations we rather prefer the LSTM architecture. At the current stage we rather see this as a proof of concept, because we have not really fine-tuned the network architectures, nor has the SGD fitting been perfected, e.g., often bigger architectures are used in combination with drop-outs, etc. We refrain from doing so, here, but refer to the relevant literature Perla et al. [301], Schnürch–Korn [330] and Wang et al. [375] for a more sophisticated fine-tuning.

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.