A Details on Neural Networks

A.1 Feed-Forward Neural Networks

We summarize the approach of Richman and Wüthrich [2021]. An FFNN can be defined as a collection of functions, so-called layers,

$$\phi^l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}, \mathbf{Z}^{l-1} \mapsto \mathbf{Z}^l := \sigma^l \left(\mathbf{W}^l \mathbf{Z}^{l-1} + b^l \right), l = 1, \ldots, L. \tag{1}$$

Here, $\mathbf{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$ is a weight matrix and $b^l \in \mathbb{R}^{n_l}$ is a bias vector, which the network tries to adjust during the calibration (training) in such a way that the resulting output minimizes some loss function such as the mean squared error, and $\sigma^l : \mathbb{R} \to \mathbb{R}$ are activation functions, which are applied element-wise and are often nonlinear. The output of an FFNN corresponding to an input $\mathbf{Z}^0$ is obtained by the concatenation of all layers,

$$\mathbf{Z}^L = \left(\phi^L \circ \cdots \circ \phi^1 \right) \left(\mathbf{Z}^0 \right). \tag{2}$$

The features we are given to build $\mathbf{Z}^0$ are year t, age x, country c and gender g, from which we predict the corresponding death rates $m^i_{x,t}$, where $i = (c, g)$. As stated in (1), we can only directly feed *numerical* inputs into a neural network, which is certainly problematic for country and gender and also for age as we decide to interpret it as a categorical rather than a numerical variable. We use embeddings [Guo and Berkhahn, 2016] to deal with this issue. An embedding is a function $e_{\mathcal{P}} : \mathcal{P} \to \mathbb{R}^{n_{\mathcal{P}}}$ which maps the set $\mathcal{P}$ of categorical values of a feature into a Euclidean space of typically low dimension $n_{\mathcal{P}}$. The conceptually intriguing aspect of embeddings is that the function $e_{\mathcal{P}}$ can be learned by the neural network itself, providing it with an opportunity to create a representation of the categorical feature which is optimal for solving the given prediction task.

Table 1 gives an overview of hyperparameters for the FFNN and the values we have tried during the cross validation procedure for hyperparameter selection. We have run cross validation on over 45000 hyperparameter combinations. In particular, we have considered variants of the MSE and MAE loss functions with age-specific weights (WMSE and WMAE), where we have used as weights for each age x the ratio of the average death rate at the lowest age x_1 to the average death rate at age x calculated on the training set. As death rates typically increase with age, this results in higher weights for lower ages and is intended to counterbalance the increasing influence of death rates at higher ages on the model calibration.

Table 1: Considered hyperparameter values of feed-forward neural networks (FFNN).

parameter	explanation	values
optimizer	numerical optimization algorithm to train the FFNN (variant of stochastic gradient descent)	Adam [Kingma and Ba, 2014], rmsprop [Tieleman and Hinton, 2012]
learning rate	step size of the gradient descent optimizer	0.001, 0.01, 0.1
batch size	to achieve faster convergence, the training data is not put into the algorithm all at once but in batches of this size	32, 128
loss function	metric to be optimized during the training	(W)MSE, MAE
number of epochs	number of times the training data are consecutively put into the optimization algorithm	50, 100, 300
early stopping	whether to abort training when there is no loss function decrease after 2 consecutive epochs	with, without

transformation of outputs	transformation to apply to the output death rates	logarithm, none
scaling of numerical inputs	neural networks require inputs to be in a similar order of magnitude [see LeCun et al., 1998]	standardization, min-max, none
embed age	whether to embed age or to treat it as a numerical variable	embed, do not embed
age embedding size	dimension of age embedding space	2, 3, 5, 7, 10
gender embedding size	dimension of gender embedding space	2, 3, 5
country embedding size	dimension of country embedding space	2, 3, 5, 7, 10
activation of hidden layers	activation function of hidden layers	relu, tanh
activation of output layer	activation function of last layer; id for log death rates, relu or sigmoid for raw death rates	id, relu, sigmoid
architecture	number of hidden layers and number of neurons per hidden layer (e.g., $(16) * 3$ means 3 hidden layers with 16 neurons each)	(16), (32), (64), (128), $(16, 8)$, $(16) * 2$, $(32) * 2$, $(64) * 2$, $(16) * 3$, $(32) * 3$, $(64) * 3$, $(128) * 5$
skip connection	whether to add a skip connection from the input to the last hidden layer [see Richman and Wüthrich, 2021]	with, without
dropout probability	regularization technique, drop weights during training with specified probability, applied after every hidden layer [see Srivastava et al., 2014]	0 (i.e., no dropout), 0.05
batch normalization	regularization technique to prevent internal covariate shift, applied after every hidden layer [see Ioffe and Szegedy, 2015]	with, without

Based on the cross validation results, we have chosen the following hyperparameter values for the FFNN:

- Adam optimizer with learning rate 0.001,
- batch size of 128 and 500 epochs without early stopping,
- minimize MAE loss in predicting raw death rates,
- use categorical features country, gender and age embedded into Euclidean spaces of dimension 3, 2 and 5, respectively, and standardized numerical feature year,
- two hidden layers with relu activations and 64 neurons each, including dropout with a probability of 0.05 after every hidden layer as well as a skip connection from the input to the second hidden layer,
- sigmoid activation for the output layer.

A.2 Recurrent Neural Networks

We begin by fixing $\tau \in \mathbb{N}$, the maximal length of a historical time window influencing current predictions, and as for CNN set $\tau = 10$.

Then, for all populations $i \in \mathcal{P}$ and all target ages $x_T \in \mathcal{X}_{\text{out}}$, we arrange the available death rates in matrices via the rolling window approach

$$\left(m^i_{x,t}\right)^\top_{x=x_T-A_P,\ldots,x_T+A_P,t=t_1,\ldots,t_\tau}, \ldots, \left(m^i_{x,t}\right)^\top_{x=x_T-A_P,\ldots,x_T+A_P,t=t_{Y-\tau},t_{Y-1}}, \quad (3)$$

where $A_P \in \mathbb{N}_0$ and a value $A_P > 0$ allows the network to use observed death rates of neighboring ages as additional features. Based on these inputs, the goal is to obtain one-step forecasts of $m^i_{x_T,t_{\tau+1}}, \ldots, m^i_{x_T,t_Y}$, respectively. In other words, an input of the network is a matrix of size $(2A_P + 1) \times \tau$, whose columns contain time series of death rates for the target age x_T and possibly some neighboring ages. Additionally, age, gender and country are input via embedding layers. We obtain forecasts for multiple years ahead by recursive one-year predictions, analogously as for CNN.

Table 2 gives an overview of hyperparameters for the RNN and the values we have tried during the cross validation procedure for hyperparameter selection. We have run cross validation on over 2000 hyperparameter combinations.

Table 2: Considered hyperparameter values of recurrent neural networks (RNN).

parameter	explanation	values
optimizer	numerical optimization algorithm to train the RNN (variant of stochastic gradient descent)	Adam [Kingma and Ba, 2014]
learning rate	step size of the gradient descent optimizer	0.001, 0.01
batch size	to achieve faster convergence, the training data is not put into the algorithm all at once but in batches of this size	10, 20, 40
loss function	metric to be optimized during the training	MSE, MAE
number of epochs	number of times the training data are consecutively put into the optimization algorithm	100, 250, 500
early stopping	whether to abort training when there is no loss function decrease after 2 consecutive epochs	with, without
transformation of outputs	transformation to apply to the output death rates	logarithm or none
scaling of numerical inputs	neural networks require inputs to be in a similar order of magnitude [see LeCun et al., 1998]	standardization, min-max
number of neighboring ages A_P	number of neighboring ages to add on each side of the target age	0, 1, 2
age embedding size	dimension of age embedding space	3, 5, 10
gender embedding size	dimension of gender embedding space	2
country embedding size	dimension of country embedding space	3, 5
activation of LSTM	activation function of LSTM layers	relu, tanh
activation of output layer	activation function of last layer; id for log death rates, sigmoid for raw death rates	id, sigmoid
architecture	number of LSTM layers and number of neurons per LSTM layer	(5), (10), (20), $(5,5)$, $(10,10)$, $(20,20)$, $(5,5,5)$

Based on the cross validation results, we have chosen the following hyperparameters for the RNN:

- Adam optimizer with learning rate 0.001,
- batch size of 40 and 500 epochs without early stopping,
- minimize MAE loss in predicting logarithmic death rates,
- use categorical features country, gender and age embedded into Euclidean spaces of dimension 5, 2 and 10, respectively, and standardized numerical feature year,
- $A_P = 2$ neighboring ages as additional inputs on each side of the target age,

- one LSTM layer with relu activation and 10 neurons.

A.3 Convolutional Neural Networks

In the following, we give a detailed description of the different components of a CNN. For this, it is convenient to interpret their input matrices

$$\mathbf{Z}^0 := (m_{x,t})_{x,t} \in \mathbb{R}^{A_{\text{in}} \times \tau}$$

as 3-hypermatrices, i.e., elements of $\mathbb{R}^{1 \times A_{\text{in}} \times \tau}$, which is isomorphic to $\mathbb{R}^{A_{\text{in}} \times \tau}$. Now, we can define a CNN as a collection $\phi^1, \ldots, \phi^L$ of functions

$$\phi^l : \mathbb{R}^{n_1^{l-1} \times n_2^{l-1} \times n_3^{l-1}} \to \mathbb{R}^{n_1^l \times n_2^l \times n_3^l} \text{ for } l = 1, \ldots, L,$$

with $n_1^l, n_2^l, n_3^l \in \mathbb{N}$, where n_2^{l-1} and n_3^{l-1} are interpreted as the numbers of rows and columns of the input matrices of layer l, and n_1^{l-1} is the number of such input matrices, also called the number of *channels*. For example, if the inputs are RGB images, $n_1^0 = 3$ could account for the three different color channels, but as we have two-dimensional input data without any natural additional channels in our application, we set $n_1^0 = 1$. Recursively defining $\mathbf{Z}^l := \phi^l\left(\mathbf{Z}^{l-1}\right)$ for $l = 1, \ldots, L$, we call

$$\left(\mathbf{Z}^l_{k,i,j}\right)_{i=1,\ldots,n_2^l,\, j=1,\ldots,n_3^l} \text{ for } k = 1, \ldots, n_1^l$$

feature maps if l is a convolutional or pooling layer (see below), and the number of these feature maps is given by n_1^l. They can be interpreted as automatically generated features, where we allow the net to create multiple feature maps per layer so that each feature map can focus on different characteristics of the input, for example edges and corners in image data. A visualization of such feature maps is given in Section C.1.

There are mainly three types of layers, i.e., kinds of functions ϕ^l, in CNN.

Convolutional layers If layer l is a convolutional layer, there is a collection of associated weights or *filters* $\mathbf{W}^l \in \mathbb{R}^{n_1^l \times n_1^{l-1} \times f \times f}$. Usually, the number of filters n_1^l per input channel and the filter size $f \in \mathbb{N}$ are specified during the implementation of the CNN. Both quantities can in principle vary across convolutional layers, but we consider only constant numbers of filters and filter sizes for simplicity, i.e., we do not let f or $n_1^l =: n_F$ depend on l. The output $\mathbf{Z}^l$ of layer l is obtained via

$$\mathbf{Z}^l_{k,i,j} = \sigma^l\left(\sum_{r=1}^{n_1^{l-1}} \sum_{p=1}^{f} \sum_{q=1}^{f} \mathbf{W}^l_{k,r,p,q} \mathbf{Z}^{l-1}_{r,i+p-1,j+q-1} + b_k\right) \tag{4}$$

for $k = 1, \ldots, n_1^l, i = 1, \ldots, n_2^l, j = 1, \ldots, n_3^l$.

Here $b \in \mathbb{R}^{n_F}$ is a bias vector, and $\sigma^l : \mathbb{R} \to \mathbb{R}$ is an activation function. For convolutional layers, we always use the rectified linear unit $\text{relu}(x) := \max(0, x)$ as the activation function.

Equation (4) can be interpreted in the following way: For every output channel $k = 1, \ldots, n_1^l$, we calculate a sliding dot product of the filter

$$\left(\mathbf{W}^l_{k,r,p,q}\right)_{r=1,\ldots,n_F,\, p,q=1,\ldots,f}$$

with the output $\mathbf{Z}^{l-1}$ of the previous layer to obtain the k-th feature map of layer l. Here, we use different filters for each output channel k, but the filter weights are shared per feature map. In this way, the CNN is enabled to detect the same features at different locations of the input data [see LeCun et al., 1989], and this also saves many parameters in comparison to a fully connected approach where different weights are trained for every possible connection. Generalizations of Equation (4) exist where the so-called *stride* is larger than 1, which means that the filters are not shifted only one but multiple units at a time in the calculation of the sliding dot product. As the size of the output data decreases with the stride and we already have rather small input data to begin with, we do not make use of this technique.

Equation (4) imposes the following relationships between the tuples $\left(n_2^l, n_3^l\right)$ and $\left(n_2^{l-1}, n_3^{l-1}\right)$:

$$n_2^l = n_2^{l-1} - f + 1, n_3^l = n_3^{l-1} - f + 1. \tag{5}$$

In particular, this implies that a convolutional layer reduces the second and third dimension of its input by $f - 1$. This could be prevented by *padding* the outputs of the convolutional layers, i.e., appending zero values at the edges such that $n_2^l = n_2^{l-1}$ and $n_3^l = n_3^{l-1}$, but we refrain from doing so as we do not consider it necessary here.

Pooling layers Pooling layers usually directly follow after convolutional layers. They are used for downsampling, i.e., for dimension reduction of feature maps. This makes sense since the calculation of multiple feature maps from the same input data point introduces some redundancy, which is then reduced by the pooling operation whose aim is to extract the most dominant output signals of the convolutional layer. Similarly to convolutional filters, pooling "filters" are slided over the feature maps. However, they do not calculate a dot product with each section of the feature map but instead a real-valued function of it. The most popular choices are *max pooling* and *average pooling*:

$$\mathbf{Z}_{k,i,j}^l = \begin{cases} \max\limits_{\substack{p=(i-1)n_P+1,\ldots,i\cdot n_P,\\ q=(j-1)n_P+1,\ldots,j\cdot n_P}} \mathbf{Z}_{k,p,q}^{l-1} & \text{(max pooling)}, \\ \frac{1}{(n_P)^2} \sum\limits_{p=(i-1)n_P+1}^{i\cdot n_P} \sum\limits_{q=(j-1)n_P+1}^{j\cdot n_P} \mathbf{Z}_{k,p,q}^{l-1} & \text{(average pooling)}, \end{cases} \tag{6}$$

for $k = 1, \ldots, n_1^l$. Here, n_P is the pooling size which determines how many units in a feature map are pooled into a single number. For example, if we perform max pooling with $n_P = 2$, this amounts to calculating a sliding maximum over groups of $2^2 = 4$ units in the input feature maps. Equation (6) fixes the dimensions of the output:

$$n_1^l = n_1^{l-1}, n_2^l = \left\lfloor \frac{n_2^{l-1}}{n_P} \right\rfloor \text{ and } n_3^l = \left\lfloor \frac{n_3^{l-1}}{n_P} \right\rfloor, \tag{7}$$

where $\lfloor z \rfloor := \max\{r \in \mathbb{N} : r \leq z\}$.

Dense layers FFNN consist exclusively of dense layers. In this sense, we can interpret the convolutional and pooling layers of a CNN as sophisticated feature extractors, from which the resulting features are passed on to a shallow FFNN which learns how to translate these features into mortality rate predictions. As dense layers expect ordinary vectors as input, the first step for training a dense layer l whose predecessor $l - 1$ is a convolutional or pooling layer consists in flattening, i.e., stacking all components of

$\mathbf{Z}^{l-1}$ into the first dimension so that we can interpret it as a column vector $\mathbf{Z}^{l-1} \in \mathbb{R}^{n^{l-1}}$ where $n^{l-1} := n_1^{l-1} \cdot n_2^{l-1} \cdot n_3^{l-1}$. The output of the dense layer is obtained via

$$\mathbf{Z}^l = \sigma^l \left(\mathbf{W}^l \mathbf{Z}^{l-1} + b^l \right), \tag{8}$$

where $\mathbf{W}^l \in \mathbb{R}^{n^l \times n^{l-1}}$ is a weight matrix, $b^l \in \mathbb{R}^{n^l}$ a bias vector and the activation function $\sigma^l : \mathbb{R} \to \mathbb{R}$ is applied element-wise. As we do not consider architectures where convolutional or pooling layers follow dense layers, we can again interpret $\mathbf{Z}^l$ as a column vector in $\mathbb{R}^{n^l}$. Also note that at least the last layer ($l = L$) in our CNN should be a dense layer because we need a column vector in $\mathbb{R}^{A_{\text{out}}}$, which contains predictions of the age-specific death rates in the following year, as an output.

Table 3 gives an overview of hyperparameters for the CNN and the values we have tried during the cross validation procedure for hyperparameter selection.

Table 3: Considered hyperparameter values of convolutional neural networks (CNN).

parameter	explanation	values
optimizer	numerical optimization algorithm to train the CNN (variant of stochastic gradient descent)	Adam [Kingma and Ba, 2014]
learning rate	step size of the gradient descent optimizer	0.001
batch size	to achieve faster convergence, the training data is not put into the algorithm all at once but in batches of this size	$50, 100$
loss function	metric to be optimized during the training	(W)MSE, (W)MAE
number of epochs	number of times the training data are consecutively put into the optimization algorithm	100, 200, 300, 400, 500
transformation of outputs	transformation to apply to the output death rates	logarithm or none
scaling of numerical inputs	neural networks require inputs to be in a similar order of magnitude [see LeCun et al., 1998]	standardization
architecture	what layers to use and in what order; c = convolutional, p = pooling, d = dense; the final output layer has to be a dense layer and is therefore not explicitly mentioned here	cp, cpd, cpdd, ccp, ccpd, cpcp, cpcpd
number of filters n_F	number of convolutional filters	5, 10, 20
size of filters f	third and fourth dimension of the filters	2, 3
type of pooling	see (6)	max, average
pooling size n_P	determines how many units are pooled	2
activation of dense layers	activation function of dense layers; id for log death rates, softplus or sigmoid for raw death rates	id, softplus sigmoid
size of dense layers	number of neurons per hidden dense layer	5, 10, 25, 50, 100
batch normalization	regularization technique to prevent internal covariate shift, applied after every convolutional layer [see Ioffe and Szegedy, 2015]	with, without

Based on the cross validation results, we have chosen the following hyperparameters for the CNN:

- Adam optimizer with learning rate 0.001,
- batch size of 100 and 500 epochs without early stopping,
- no batch normalization,
- minimize MAE loss in predicted logarithmic death rates,

- standardize inputs,
- start with one convolutional layer with $n_F = 10$ filters of size $f = 3$ followed by an average pooling layer of size $n_P = 2$. Repeat these two layers a second time, insert a dense layer with 50 neurons and id activation and, finally, the dense output layer with 30 neurons and id activation.

B Prediction Intervals for Stochastic Mortality Models

We describe how we calculate prediction intervals in the Poisson LC model, which is based on the equation

$$\log m^i_{x,t} = \alpha^i_x + \beta^i_x \kappa^i_t. \tag{9}$$

Usually, ARIMA models are used to extrapolate the calibrated period effect $\left(\hat{\kappa}^i_t\right)_t$, with the most common case being just a random walk with drift:

$$\hat{\kappa}^i_{t+1} = d^i + \hat{\kappa}^i_t + \varepsilon^i_{t+1} \text{ with } d^i \in \mathbb{R} \text{ and } \varepsilon^i_{t+1} \sim \mathcal{N}\left(0, \left(\sigma^i_\varepsilon\right)^2\right) \text{ i.i.d.} \tag{10}$$

Its parameters are estimated by

$$\hat{d}^i := \frac{\hat{\kappa}^i_{t_Y} - \hat{\kappa}^i_{t_1}}{Y-1} \text{ and } \left(\hat{\sigma}^i_\varepsilon\right)^2 := \frac{1}{Y-2} \sum_{t=t_2}^{t_Y} \left(\hat{\kappa}^i_t - \hat{\kappa}^i_{t-1} - \hat{d}^i\right)^2. \tag{11}$$

Forecasts for future death rates are obtained by inserting the central projection of the random walk into the death rate model. More precisely, we set

$$\hat{m}^i_{x,t_Y+h} := \exp\left(\hat{\alpha}^i_x + \hat{\beta}^i_x \left(\hat{\kappa}^i_{t_Y} + h \cdot \hat{d}^i\right)\right) \tag{12}$$

for $h \in \mathbb{N}$ [see Koissi et al., 2006]. Similarly, prediction interval bounds at level $a \in (0,1)$ for the death rate m^i_{x,t_Y+h} could be obtained as

$$\exp\left(\hat{\alpha}^i_x + \hat{\beta}^i_x \left(\hat{\kappa}^i_{t_Y} + h \cdot \hat{d}^i \pm \sqrt{h} \cdot \hat{\sigma}^i_\varepsilon \cdot \Phi^{-1}\left(\frac{1+a}{2}\right)\right)\right),$$

where Φ^{-1} denotes the quantile function of the standard normal distribution.

However, these bounds only take into account the random volatility of the time series, i.e., noise variance but not the uncertainty in estimating the ARIMA parameters, i.e., model uncertainty. In the simple random walk with drift case, this is just the uncertainty related to the estimation of the drift (there is also uncertainty in the estimation of the time series volatility itself, which is neglected here). To account for this, we define the prediction interval bounds via

$$\hat{m}^{i,\,\text{lower|upper}}_{x,t_Y+h} := \exp\left(\hat{\alpha}^i_x + \hat{\beta}^i_x \left(\hat{\kappa}^i_{t_Y} + h \cdot \hat{d}^i \pm \hat{s} \cdot \Phi^{-1}\left(\frac{1+a}{2}\right)\right)\right), \tag{13}$$

with

$$\hat{s} := \sqrt{h^2 \cdot \hat{\sigma}^2_{\hat{d}^i} + h \cdot \left(\hat{\sigma}^i_\varepsilon\right)^2}, \tag{14}$$

where $\hat{\sigma}^2_{\hat{d}^i}$ is the empirical variance of the estimator $\hat{d}^i$, which is given by

$$\hat{\sigma}^2_{\hat{d}^i} := \frac{\left(\hat{\sigma}^i_\varepsilon\right)^2}{Y-1}. \tag{15}$$

For a more detailed derivation and for the estimation of prediction intervals in the general ARIMA model, we refer to Kleinow and Richards [2016].

While there are other approaches such as bootstrapping [Koissi et al., 2006] to account for further sources of uncertainty in the calibration of the LC model [for a detailed discussion, see also Lee and Carter, 1992], we use (13) as it accounts for the uncertainty involved in the actual forecasting process. Furthermore, we assume it is applied most often in practice.

There is no closed formula for the prediction interval bounds of the ACF model, so they are calculated via Monte Carlo simulation (ignoring parameter uncertainty).

C Additional Empirical Results

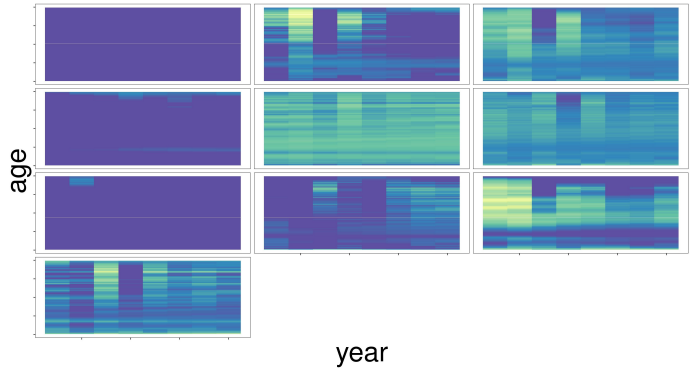
C.1 Plotting Feature Maps

A technique which is often used to visualize CNN and to convince oneself that the feature extraction performed by the convolutional layers makes sense consists in plotting the feature maps for certain inputs. For images, those often look like variations of the input showing where the network places its focus with the different filters. While the interpretation for the human eye is easier in the case of images, it is possible to make some observations for mortality data. In Figure 1, we display the ten feature maps of the first convolutional layer of the first model in the CNN ensemble to which we have input the death rates of English & Welsh females in the time periods 1891–1900, 1941–1950 and 1991–2000. We can clearly see that different filters focus on different parts of the age-year input matrices, and that it also depends on the time period (i.e., on the level of the death rates) whether filters are activated more or less. In particular, some filters are not activated at all for certain time periods, which is due to the relu activation function used in the convolutional layers. There are some diagonal patterns in the feature maps, most clearly visible for the years 1991–2000, indicating that the network has detected cohort effects. In particular, this proves the greater flexibility of the CNN in accounting for changing dependencies of age and year, while in the LC model the influence of yearly changes in mortality on different ages represented by the age effect stays constant over time.

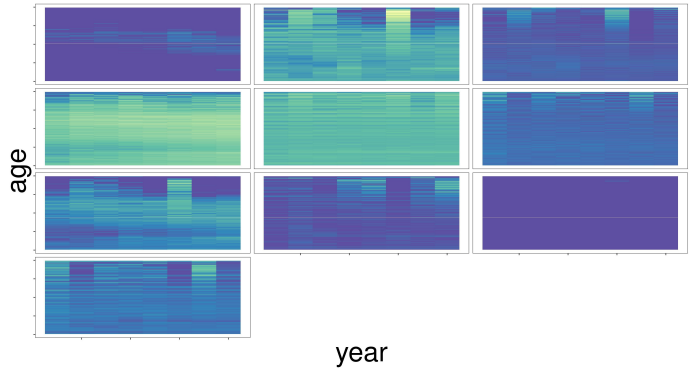
C.2 Evaluation over the Whole Age Range

Table 4 displays the performance of the different models over the whole age range 2–98. The youngest and oldest ages 0, 1, 99 and 100 are excluded for technical reasons (the RNN needs additional age columns on each side of the target age and these are not available for the youngest and oldest ages). Furthermore, we exclude Iceland from the evaluation because there are calibration issues with the LC model for this country.

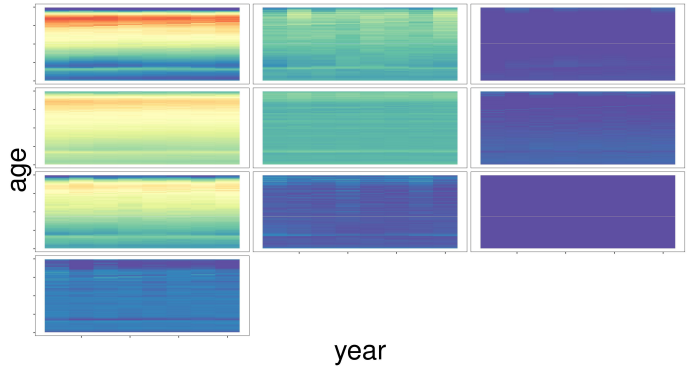
The results are broadly similar to those of Tables 2 and 4 from the main paper. CNN and FFNN yield similar MSE and MAE as well as a PICP above 95%. The MdAPE is clearly minimized by the CNN. The MPIW of FFNN and RNN are heavily inflated by a few populations with overestimated noise variance. The MPIW of CNN is significantly larger than those of LC and ACF, which also indicates some inflation, but more stable compared to FFNN and RNN. Therefore, even though FFNN gets closer to the target coverage probability of 95%, CNN prediction intervals would be preferable.



(a) 1891–1900



(b) 1941–1950



(c) 1991–2000

Figure 1: Feature maps of the first convolutional layer for English & Welsh females for input years 1891–1900, 1941–1950 and 1991–2000 (blue means no activation, red means high activation).

Table 4: Robustness check. Out-of-sample error measures for 52 populations, ages 2 to 98, years 2007 to 2016 (models trained on years up to 2006). In case of the RNN, the female populations of Northern Ireland, Slovenia and Scotland were not included in the evaluation due to numerical instabilities. The best value in each column, where applicable, is marked in bold.

Model	MSE$\times 10^4$	MAE$\times 10^3$	MdAPE[%]	Dev	PICP[%]	MPIW
LC10	2.4	4.0	10.1	**12.3**	50.9	0.0079
LC20	1.8	4.1	10.3	16.5	50.1	0.0064
ACF	1.8	4.3	11.6	17.2	57.2	0.0087
FFNN	1.3	3.1	11.3	34.1	**96.4**	$\gg 1$
RNN	1.4	3.6	9.5	13.2	45.3	$\gg 1$
CNN	**1.2**	**3.0**	**8.8**	17.4	97.2	0.2662

C.3 Annuity Values

We consider a temporary annuity-immediate issued to a life of population i aged x at the start of year t, which runs for n years and pays an amount of 1 at the end of each year in which the life is alive. Its present value, assuming a constant yearly discount factor v, is given by

$$\sum_{s=1}^{n} v^s \, {}_sp^i_{x,t} = \sum_{s=1}^{n} v^s \prod_{j=0}^{s-1} p^i_{x+j,t+j} \approx \sum_{s=1}^{n} v^s \exp\left(- \sum_{j=0}^{s-1} m^i_{x+j,t+j} \right), \tag{16}$$

where we have used the notation ${}_sp^i_{x,t}$ for the s-year cohort survival probability with special case $p^i_{x,t} := {}_1p^i_{x,t}$ and the common approximation $p^i_{x,t} \approx \exp\left(-m^i_{x,t}\right)$. This approximation is based on assuming a piecewise constant force of mortality between integer ages, which is unlikely to hold for higher ages. Other approximation formulae are possible but would not result in substantial qualitative changes of the following results.

Figure 2 displays the values of (16) according to the different models for $x = 60$, $t = 2007$, $n = 30$ and $v = \frac{1}{1.009}$. We observe large differences between populations, which are expected due to the differing levels of mortality, but for some populations such as Lithuanian males also noticeable differences between models.

For the female and male populations of West Germany, we additionally compare the annuity values predicted by the models to those implied by the second-order annuity life tables DAV2004R provided by the German Association of Actuaries [DAV, 2018], see Table 5. To get an impression of the uncertainty in the central forecasts, we provide the annuity values obtained from both the central death rate estimates and the 95% prediction interval bounds. Note that low annuity values correspond to high death rates, which means that the annuity value lower bounds are calculated from the death rate upper bounds, and vice versa. We find that the actuarial life table leads to significantly more conservative reserves for both genders than the central estimates of our considered mortality models, while the annuity value upper bounds resulting from the prediction intervals are closer to the values implied by DAV2004R. Again, there are also non-negligible differences in the values implied by our models which range from 21.3 to 22.3 (central estimates) or 20.0 to 23.0 (prediction intervals) for females and from 19.2 to 19.7 (central estimates) or 17.6 to 21.0 (prediction intervals) for males. In practical pricing

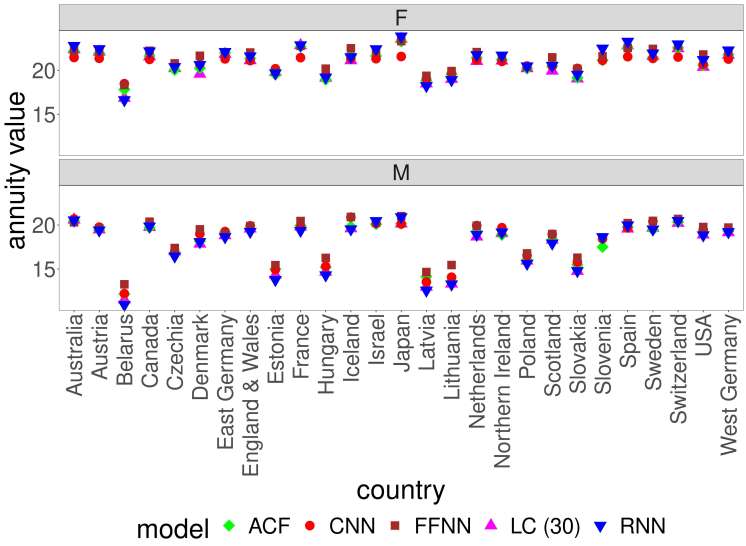


Figure 2: Population-specific values of a 30-year annuity for a life aged $x = 60$ at the start of year $t = 2007$ according to CNN (red circles), FFNN (brown squares), RNN (blue inverted triangles), ACF (green diamonds) and LC30 (magenta triangles) under a yearly discount factor of $v = \frac{1}{1.009}$.

or reserving applications, it is therefore recommendable to consult multiple models and compare the resulting projections, annuity values and their lower and, more importantly, upper bounds with careful actuarial judgement.

Table 5: Values of a 30-year annuity for a female or male life aged $x = 60$ at the start of year $t = 2007$ in West Germany according to the 95% prediction intervals and central estimates of different models and the DAV2004R life table.

Model		LC30	ACF	CNN	FFNN	RNN	2004R
Annuity value (F)	lower bound	21.1	21.3	20.0	21.3	21.4	—
	central estimate	21.7	21.8	21.3	22.1	22.3	23.0
	upper bound	22.3	22.2	22.3	22.8	23.0	—
Annuity value (M)	lower bound	18.5	18.7	17.6	18.7	18.4	—
	central estimate	19.2	19.3	19.5	19.7	19.2	21.2
	upper bound	19.8	19.9	21.0	20.6	20.0	—