Convolutional Neural Networks

Mario V. Wüthrich
RiskLab, ETH Zurich

RiskLab®
Switzerland

"Deep Learning with Actuarial Applications in R"
Swiss Association of Actuaries SAA/SAV, Zurich
October 14/15, 2021

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:30)
- Feed-Forward Neural Networks (THU 13:00-15:00)
- Discrimination-Free Insurance Pricing (THU 17:15-17:45)

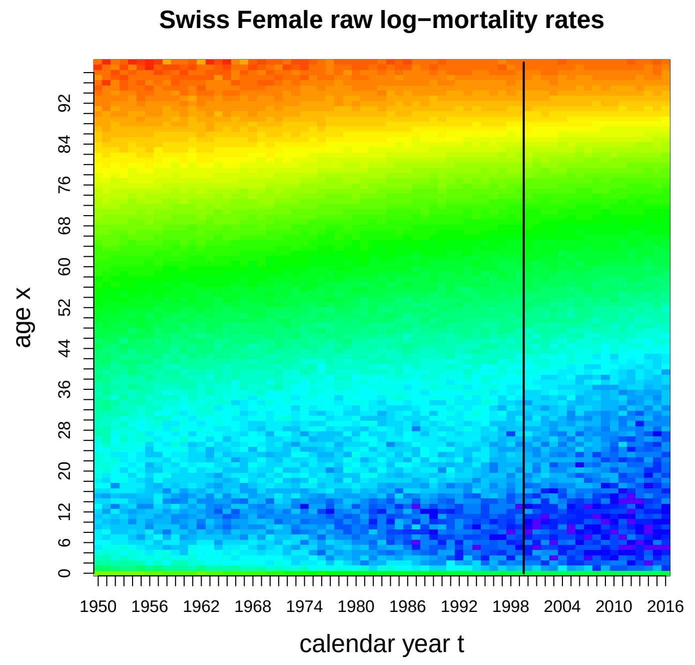
- LocalGLMnet (FRI 9:00-10:30)
- Convolutional Neural Networks (FRI 13:00-14:30)
- Wrap Up (FRI 16:00-16:30)

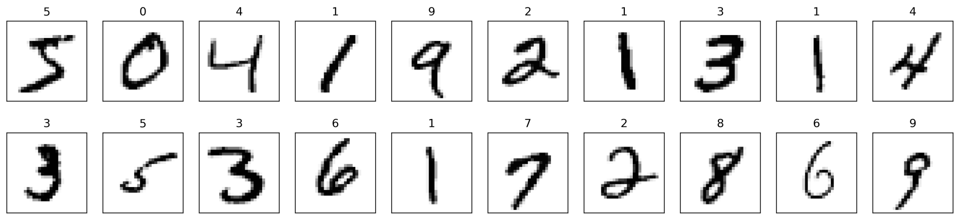
Contents: Convolutional Neural Networks

- Spatial and temporal data
- Convolutional neural networks (CNNs)
- Special purpose tools for CNNs
- CNN examples

- **Spatial and Temporal Data**

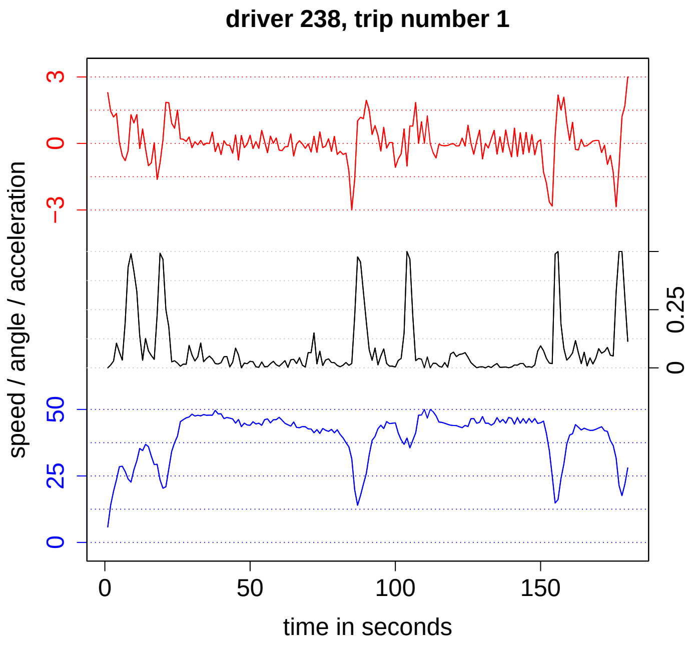
Spatial Objects





- Spatial objects are tensors $\boldsymbol{Z} \in \mathbb{R}^{I \times J \times q}$ of order/mode 3, i.e. 3-dimensional arrays.
- The first two components of $\boldsymbol{Z}$ give the location (i, j) in the picture.
- The 3rd component of $\boldsymbol{Z}$ gives the signals in location (i, j). This 3rd component is called channels. Black-white pictures have $q = 1$ channel (gray scale), color pictures have $q = 3$ channels (RGB channels for red-green-blue).

Temporal and Time Series Objects



'storm damaged controller at courthouse'

- Temporal objects are matrices $\boldsymbol{Z} \in \mathbb{R}^{T \times q}$ which are tensors of order/mode 2.
- The 1st component of $\boldsymbol{Z}$ gives the location t in the time series.
- The 2nd component of $\boldsymbol{Z}$ gives the signals in location t having q channels.

Using FNNs for Time Series Processing

- Assume we have time series information $\boldsymbol{Z} = \boldsymbol{x}_{0:T} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)^\top \in \mathbb{R}^{(T+1)\times q}$ to predict response Y_{T+1}

$$\boldsymbol{x}_{0:T} \;\mapsto\; \mu_{T+1}(\boldsymbol{x}_{0:T}) = \mathbb{E}[Y_{T+1}|\boldsymbol{x}_{0:T}] = \mathbb{E}[Y_{T+1}|\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T].$$

- In principle, we could choose a FNN architecture and set

$$\mu_{T+1}(\boldsymbol{x}_{0:T}) = g^{-1}\left(\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_{0:T})\right\rangle\right).$$

- This is not recommended:
  - ⋆ Whenever we collect a new observation $\boldsymbol{x}_{T+1}$ we need to extend the input dimension of the FNN and re-calibrate it.
  - ⋆ The FNN does not recognize any temporal (topological) structure.
  - ⋆ The latter also applies to spatial objects.

RNNs, CNNs and Attention Layers

- There are 3 different ways in network modeling to deal with topological data.

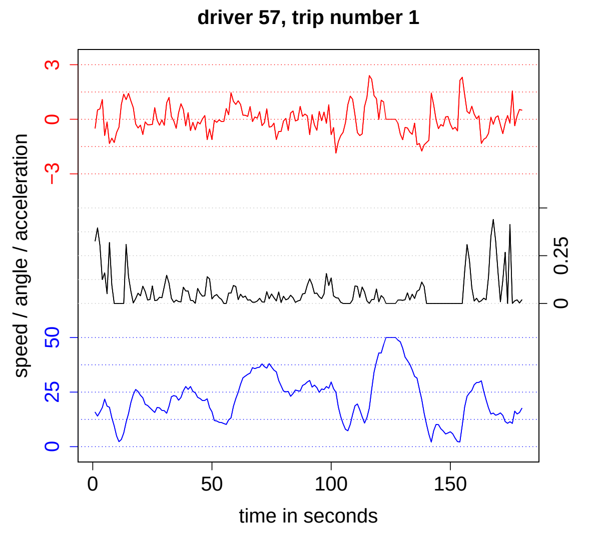
- Recurrent neural networks (RNNs) process information recursively to preserve time series structure. RNNs are most suitable to predict the next response Y_{T+1} based on past information $\boldsymbol{x}_{0:T}$.

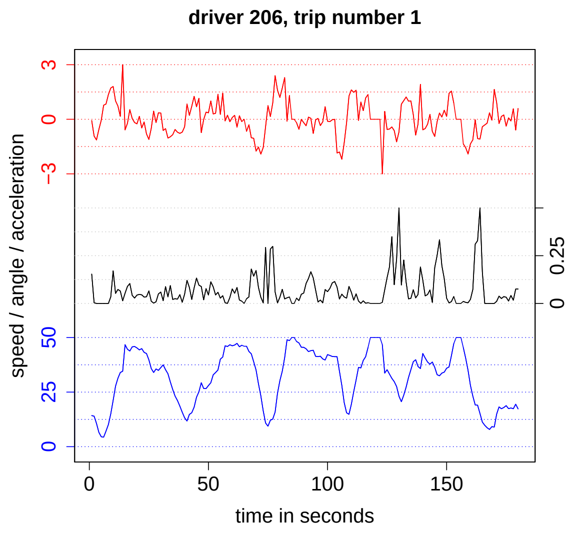
- Convolutional neural networks (CNNs) extract local structure from topological objects preserving the topology. This is done by moving small windows, say, across the picture and trying to identify specific structure in this window.
  ▷ This is similar to rolling windows in financial time series estimation.
  ▷ CNNs act more locally, whereas RNNs and FNNs act more globally.

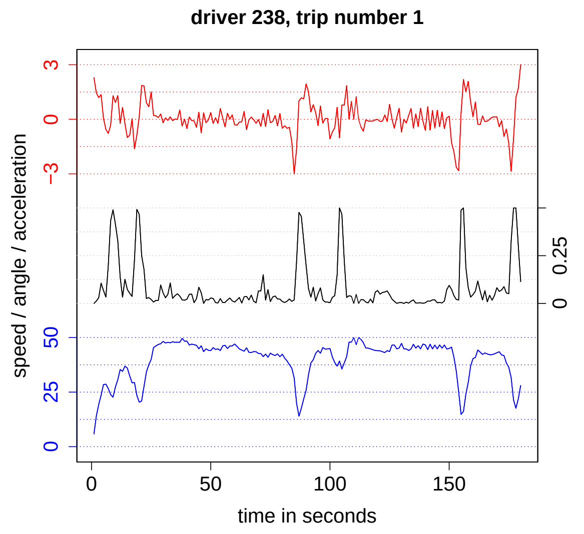
- Attention layers move across the time series and try to pay attention to special features in the time series, like giving more or less credibility to them. This is similar to the regression attentions $\boldsymbol{\beta}(\boldsymbol{x})$ in LocalGLMnets.

- **Convolutional Neural Networks (CNNs)**

Functioning of CNNs







- Choose a window (called filter), say, of size $b \times q = 10 \times 3$.
- b is called filter size, kernel size or band width; q is the number of channels.
- Move with this filter across the time series (in time direction t) and try to spot specific structure with this filter (in the rolling window).
- The way of finding structure is with a convolution operation $*$.

CNNs: More Formally

- Start from an input tensor $\boldsymbol{x} \in \mathbb{R}^{I \times J \times q_0}$ of order 3.
- Choose filter sizes $(b_1, b_2, q_0)^\top \in \mathbb{N}^3$ with $b_1 < I$ and $b_2 < J$.
- A CNN operation is a mapping

$$\begin{aligned} \boldsymbol{z}_k : \mathbb{R}^{I \times J \times q_0} &\to \mathbb{R}^{(I-b_1+1)\times(J-b_2+1)} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}_k(\boldsymbol{x}) = \left(z_{k;i,j}(\boldsymbol{x})\right)_{1\le i\le I-b_1+1;1\le j\le J-b_2+1}, \end{aligned}$$

having, for activation function $\phi : \mathbb{R} \to \mathbb{R}$,

$$z_{k;i,j}(\boldsymbol{x}) = \phi\left(w_k + \sum_{l_1=1}^{b_1}\sum_{l_2=1}^{b_2}\sum_{l_3=1}^{q_0} w_{k;l_1,l_2,l_3}\, x_{i+l_1-1,j+l_2-1,l_3}\right),$$

for given intercept $w_k \in \mathbb{R}$ and filter weights $\boldsymbol{W}_k = (w_{k;l_1,l_2,l_3})_{l_1,l_2,l_3}$.

CNNs: Convolution Operation

- Choose the corner $(i, j, 1)$ of the tensor as base point. CNN operation considers

$$(i, j, 1) + [0 : b_1 - 1] \times [0 : b_2 - 1] \times [0 : q_0 - 1],$$

  with filter weights $\boldsymbol{W}_k$.

- In fact, we perform a sort of convolution which motivates compact notation

$$\begin{aligned} \boldsymbol{z}_k : \mathbb{R}^{I \times J \times q_0} \quad &\to \quad \mathbb{R}^{(I-b_1+1)\times(J-b_2+1)} \\ \boldsymbol{x} \quad &\mapsto \quad \boldsymbol{z}_k(\boldsymbol{x}) = \phi\left(\boldsymbol{W}_k * \boldsymbol{x}\right). \end{aligned}$$

- This convolution operation $*$ reflects one filter with filter weights $\boldsymbol{W}_k$. We can now choose multiple filters (similar to neurons in FNNs):
  ▷ This explains the meaning of the lower index k (which plays the role of different neurons $1 \leq k \leq q_1$ in FNNs).

CNNs: Multiple Filters

- Choose $q_1 \in \mathbb{N}$ filters, each having filter weights $\boldsymbol{W}_k$, $1 \leq k \leq q_1$.

- A CNN layer is a mapping

$$\begin{aligned}\boldsymbol{z}^{\mathrm{CNN}} : \mathbb{R}^{I \times J \times q_0} &\to \mathbb{R}^{(I-b_1+1)\times(J-b_2+1)\times q_1}\\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\mathrm{CNN}}(\boldsymbol{x}) = (\boldsymbol{z}_1(\boldsymbol{x}), \ldots, \boldsymbol{z}_{q_1}(\boldsymbol{x})),\end{aligned}$$

  with filters $\boldsymbol{z}_k(\boldsymbol{x}) = \phi\left(\boldsymbol{W}_k * \boldsymbol{x}\right)$.

- Thus, the (spatial) tensor

$$\boldsymbol{x} \in \mathbb{R}^{I \times J \times q_0},$$

  with q_0 channels is mapped to a (spatial) tensor

$$\boldsymbol{z}^{\mathrm{CNN}}(\boldsymbol{x}) \in \mathbb{R}^{(I-b_1+1)\times(J-b_2+1)\times q_1},$$

  with q_1 filters.

Properties of CNNs

- The convolution operation $*$ respects the local structure.
- FNNs extract global structure, CNNs extract local structure.
- Formally, the global scalar product $\boldsymbol{z}_k(\boldsymbol{x}) = \phi\langle \boldsymbol{w}_k, \boldsymbol{x}\rangle$ of FNNs is replaced by a local convolution $\boldsymbol{z}_k(\boldsymbol{x}) = \phi\left(\boldsymbol{W}_k * \boldsymbol{x}\right)$ for CNNs.
- CNNs have translation invariance properties, see Wiatowski–Bölcskei (2018).
- CNNs generally use less parameters than FNNs and RNNs, because filter weights are re-used/re-located.

- **Special Purpose Tools for CNNs**

CNNs: Padding with Zeros

- A CNN layer reduces the spatial size of the output tensor

$$\begin{aligned} \boldsymbol{z}^{\text{CNN}} : \mathbb{R}^{I\times J\times q_0} &\to \mathbb{R}^{(I-b_1+1)\times(J-b_2+1)\times q_1} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\text{CNN}}(\boldsymbol{x}) = (\boldsymbol{z}_1(\boldsymbol{x}),\ldots,\boldsymbol{z}_{q_1}(\boldsymbol{x})). \end{aligned}$$

- If this is an undesired feature, padding with zeros can be applied at all edges to obtain

$$\begin{aligned} \boldsymbol{z}^{\text{CNN}} : \mathbb{R}^{I\times J\times q_0} &\to \mathbb{R}^{I\times J\times q_1} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\text{CNN}}(\boldsymbol{x}) = (\boldsymbol{z}_1(\boldsymbol{x}),\ldots,\boldsymbol{z}_{q_1}(\boldsymbol{x})). \end{aligned}$$

- Remark that padding does not add any additional parameters, but it is only used to reshape the output tensor.

CNNs: Stride

- Strides are used to skip part of the input tensor $\boldsymbol{x}$ to reduce the size of the output. This may be useful if the input tensor is a very high resolution image.

- Choose stride parameters s_1 and s_2. Consider modified convolution

$$\sum_{l_1} \sum_{l_2} \sum_{l_3} w_{k;l_1,l_2,l_3} \, x_{s_1(i-1)+l_1, s_2(j-1)+l_2, l_3}.$$

- This considers windows

$$(s_1(i-1), s_2(j-1), 1) + [1 : b_1] \times [1 : b_2] \times [0 : q_0 - 1].$$

CNNs: 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 dilation parameters e_1 and e_2. Consider modified convolution

$$\sum_{l_1} \sum_{l_2} \sum_{l_3} w_{k;l_1,l_2,l_3} \; x_{i+e_1(l_1-1),j+e_2(l_2-1),l_3}.$$

- This considers

$$(i,j,1) + e_1 \, [0 : b_1 - 1] \times e_2 \, [0 : b_2 - 1] \times [0 : q_0 - 1].$$

CNNs: Max-Pooling Layers

- Pooling layers help to reduce the sizes of the tensors.

- We choose fixed window sizes b_1 and b_2 and strides $s_1 = b_1$ and $s_2 = b_2$; this gives a partition (disjoint windows).

- The max-pooling layer then considers

$$\begin{aligned} \boldsymbol{z}^{\mathrm{Max}} : \mathbb{R}^{I\times J\times q_0} &\to \mathbb{R}^{I'\times J'\times q_0} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\mathrm{Max}}(\boldsymbol{x}) = \mathrm{MaxPool}(\boldsymbol{x}), \end{aligned}$$

  with $I' = \lfloor I/b_1 \rfloor$ and $J' = \lfloor J/b_2 \rfloor$ (cropping last columns by default), and where the convolution operation $*$ is replaced by a $\max$ operation (modulo channels).

- This extracts the maximums from the (spatially disjoint) windows

$$[b_1(i-1)+1 : b_1 i] \times [b_2(j-1)+1 : b_2 j] \times [k],$$

  for each channel $1 \le k \le q_0$, individually.

CNNs: Flatten Layers

- A flatten layer is used to reshape a tensor to a vector.
- Consider mapping

$$\begin{aligned} \boldsymbol{z}^{\text{flatten}} : \mathbb{R}^{I\times J\times q_0} &\to \mathbb{R}^{q_1} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\text{flatten}}(\boldsymbol{x}) = (x_{1,1,1}, \ldots, x_{I,J,q_0})^\top, \end{aligned}$$

  with $q_1 = I \cdot J \cdot q_0$.

- The flattened object $\boldsymbol{z}^{\text{flatten}}(\boldsymbol{x})$ can serve as input to a FNN.
- We have already met this operator with embedding layers for categorical features.

CNNs: Example (1/2)

```
library(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()
```

$$180{\times}50{\times}3 \overset{\text{CNN1}}{\mapsto} 170{\times}45{\times}10 \overset{\text{Max1}}{\mapsto} 17{\times}9{\times}10 \overset{\text{CNN2}}{\mapsto} 12{\times}6{\times}5 \overset{\text{Max2}}{\mapsto} 4{\times}3{\times}5 \overset{\text{flatten}}{\mapsto} 60.$$

CNNs: Example (2/2)

```
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
```

$$180\times50\times3 \overset{\text{CNN1}}{\mapsto} 170\times45\times10 \overset{\text{Max1}}{\mapsto} 17\times9\times10 \overset{\text{CNN2}}{\mapsto} 12\times6\times5 \overset{\text{Max2}}{\mapsto} 4\times3\times5 \overset{\text{flatten}}{\mapsto} 60.$$

- **Time Series Example: Telematics Data**

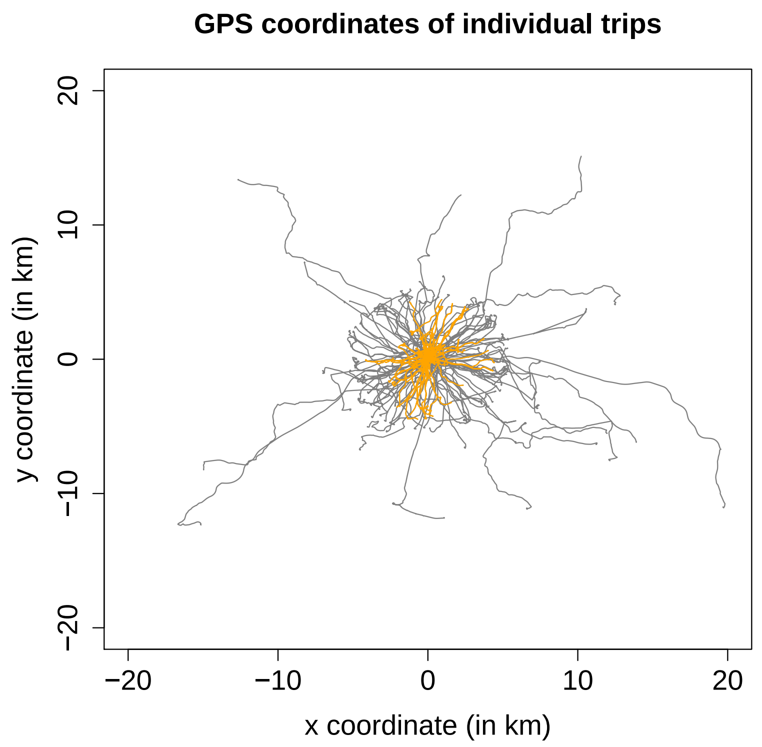
What is Telematics Car Driving Data?

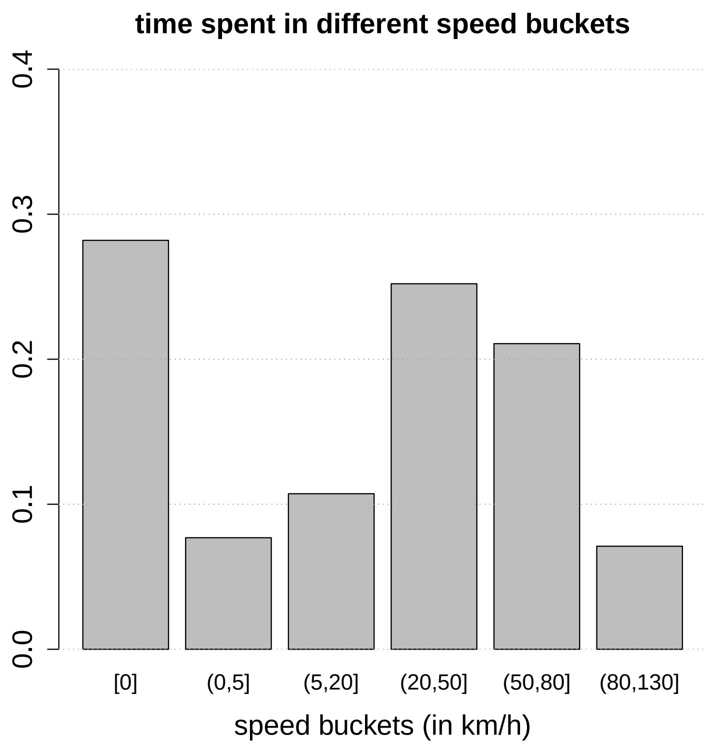
- GPS location data second by second, speed, acceleration, braking, intensity of left and right turns, engine revolutions,
- vehicle sensors and cameras,
- time stamp (day time, rush hour, night, etc.), total distances at different times,
- road type, traffic conditions, weather conditions, etc.,
- traffic rules (e.g. speeding), driving and health conditions, etc.

Volume of telematics car driving data, back of envelope calculation:

- 100KB of telematics data per driver and per day.
- This amounts to 40MB of data per driver per year.
- A small portfolio of 100'000 drivers results every year in 4TB data.

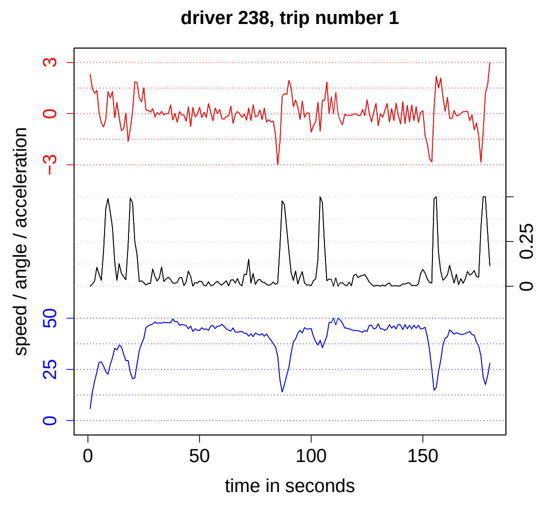
Illustration of GPS Location Data of Selected Driver





Remark that the idling phase is comparably large,
typically, we truncate the idling phase in our analysis.

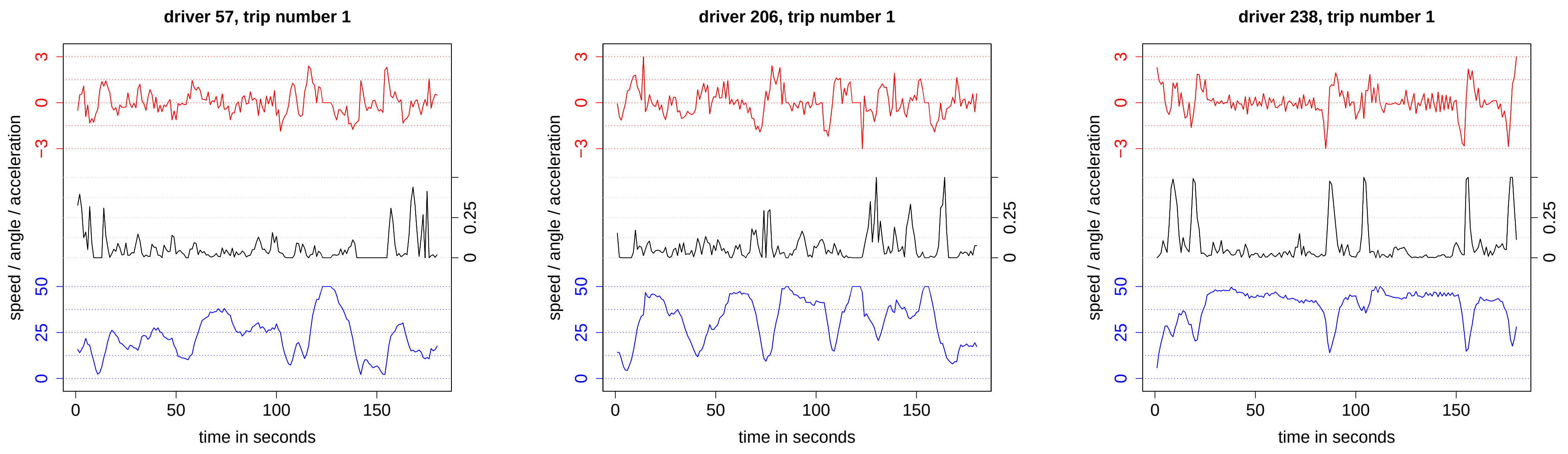
Speed, Acceleration/Braking and Direction



acceleration (m/s^2) / change in direction ($|\sin|$/s)/ speed (km/h)

- We have 3 channels:
  - ⋆ Speed v is concatenated so that $v \in [2, 50]$km/h.
  - ⋆ Acceleration a is censored at ± 3m/s^2 because of scarcity of data and data error. Extreme acceleration $+6$m/s^2, extreme deceleration -8m/s^2.
  - ⋆ Change of direction Δ is censored at $1/2$.

Choose 3 Selected Drivers



- Consider 3 selected drivers, called drivers 57, 206 and 238.
- **Question:** Can we correctly allocate individual trips to the right drivers?
- Assume that of each trip we have 180 seconds of driving experience (at random chosen from the entire trip and pre-processed as described above).

Classification with CNNs

- We choose a CNN because we would like to find similar structure in telematics time series data to discriminate the 3 different drivers.

- Consider (speed-acceleration-change in angle) time series, for $t = 1, \ldots, T = 180$,

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

where $s = 1, \ldots, S$ labels the individual trips of the considered drivers.

- Define 3-dimensional time series feature (covariate with 3 channels)

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

with categorical response $Y_s \in \{57, 206, 238\}$ indicating the drivers.

Logistic Regression for Classification

- Multinomial logistic regression uses linear predictors on the canonical scale

$$\boldsymbol{x} \mapsto \mathbf{p}^{\text{Logistic}}(\boldsymbol{x}) = \text{softmax}\left\langle \mathbf{B}, \boldsymbol{z}^{\text{flatten}}(\boldsymbol{x}) \right\rangle \in (0,1)^3,$$

  with regression parameters $\mathbf{B} \in \mathbb{R}^{180 \cdot 3 \times 3}$:

  ⋆ we need to pre-process time series feature $\boldsymbol{x} \in \mathbb{R}^{180 \times 3}$ for suitable shape (flatten to vector) and for suitable functional form (not done here);
  ⋆ the scalar product is understood column-wise in $\mathbf{B}$;
  ⋆ the softmax function is (here) for $j = 1, 2, 3$ given by

$$\text{softmax}\left\langle \mathbf{B}, \boldsymbol{z} \right\rangle_j = \frac{\exp\langle \mathbf{b}_j, \boldsymbol{z} \rangle}{\sum_{k=1}^{3} \exp\langle \mathbf{b}_k, \boldsymbol{z} \rangle} \in (0,1).$$

  where $\mathbf{b}_j$ is the j-th column of $\mathbf{B}$.

CNNs for Logistic Regression

- Multinomial logistic regression uses linear predictors on the canonical scale

$$\boldsymbol{x} \ \mapsto \ \mathbf{p}^{\text{Logistic}}(\boldsymbol{x}) \ = \ \text{softmax}\left\langle \mathbf{B}, \boldsymbol{z}^{\text{flatten}}(\boldsymbol{x}) \right\rangle \ \in \ (0,1)^3,$$

with regression parameters $\mathbf{B} \in \mathbb{R}^{180 \cdot 3 \times 3}$.

- We choose a CNN architecture of depth $d \in \mathbb{N}$ for multinomial logistic regression

$$\boldsymbol{x} \ \mapsto \ \mathbf{p}^{\text{CNN}}(\boldsymbol{x}) \ = \ \text{softmax}\left\langle \mathbf{B}, \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}) \right\rangle \ \in \ (0,1)^3,$$

with layers $\boldsymbol{z}^{(m)}$ being described on the next slide.

R Code for CNN Architecture on Time Series Data

```
model <- keras_model_sequential()

#
model %>%

layer_conv_1d(filters=12,kernel_size=5,activation='tanh',input_shape=c(180,3)) %>%
      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 = 0.3) %>%

layer_dense(units = 3, activation = 'softmax')
```

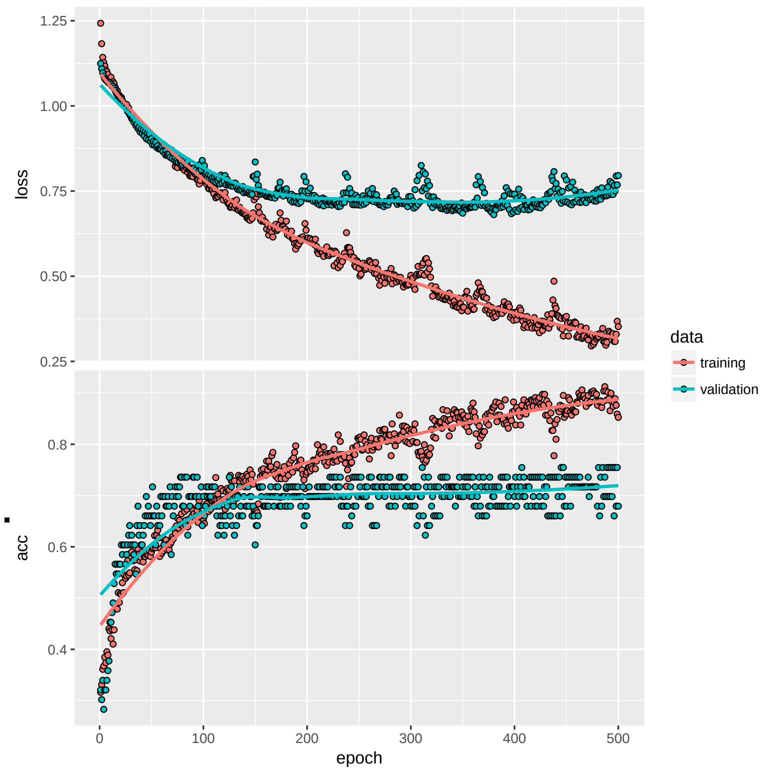
$$180 \times 3 \overset{\text{CNN1}}{\mapsto} 176 \times 12 \overset{\text{Max1}}{\mapsto} 58 \times 12 \overset{\text{CNN2}}{\mapsto} 54 \times 10 \overset{\text{Max2}}{\mapsto} 18 \times 10 \overset{\text{CNN3}}{\mapsto} 14 \times 8 \overset{\text{GlobMax}}{\mapsto} 8 \overset{\text{FNN}}{\mapsto} 3.$$

Explicit CNN Architecture and Network Parameters

```

Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 176, 12)           192
_________________________________________________________________
max_pooling1d_1 (MaxPoolin (None, 58, 12)              0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 54, 10)            610
_________________________________________________________________
max_pooling1d_2 (MaxPoolin (None, 18, 10)              0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 14, 8)             408
_________________________________________________________________
global_max_pooling1d_1 (Gl (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
```

Gradient Descent Fitting

- Total data: 521+131 individual trips $\mathcal{L}$ and $\mathcal{T}$
- We use 521 trips $\mathcal{L}$ to learn the network.
- We out-of-sample predict on 131 trips $\mathcal{T}$.
- Split 521 trips 8:2 for train/validation $\mathcal{U}$ and $\mathcal{V}$.
- Callback retrieves best network.
- Gradient descent fitting takes 40 sec.
- The plot shows deviance losses and accuracy/misclassification rate.



Out-of-sample Results

- Out-of-sample confusion matrix on $\mathcal{T}$ (131 trips):

	true labels driver 57	driver 206	driver 238
predicted label 57	33	4	0
predicted label 206	8	38	6
predicted label 238	1	5	36
% correct	78.6%	80.9%	85.7%

- This excellent prediction is based on "minimal information":
  - ⋆ only 180 seconds per trip;
  - ⋆ only very few trips to fit the network (521);
  - ⋆ not optimal data quality;
  - ⋆ not much network fine-tuning has been done.

Other Triples of Drivers

	true labels		
	driver 300	driver 301	driver 302
predicted label 300	61	1	3
predicted label 301	5	42	11
predicted label 302	8	11	25
% correct	82.4%	77.8%	65.8%

	true labels		
	driver 100	driver 150	driver 200
predicted label 100	43	12	2
predicted label 150	5	64	5
predicted label 200	4	2	51
% correct	82.7%	82.1%	87.9%

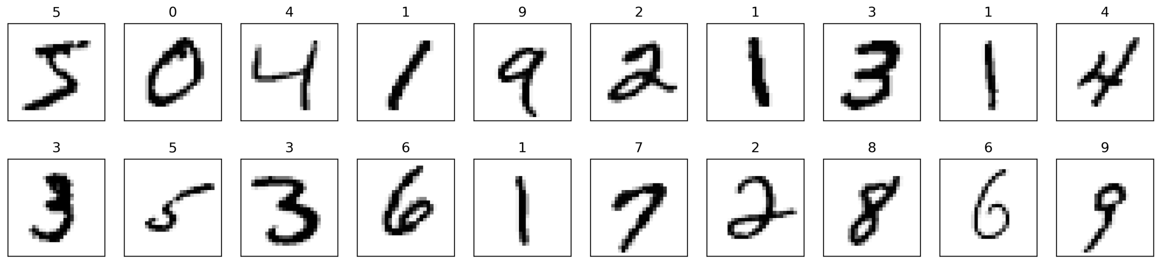
What's Next?

- Do you have any privacy concerns?

- Can we use this data to identify different driving styles?
- How much telematics data is needed to characterize a given driver?
- Can this data be made useful to improve driving behavior and style?

- **Spatial Example: Digits Recognition**

Modified National Institute of Standards and Technology (MINST) Data Set



- We have black-white pictures, i.e., 1 channel.
- Spatial objects are represented by tensors $\boldsymbol{x} \in [0,1]^{28\times 28\times 1}$ of order/mode 3.
- We have a classification problem with categorical response $Y \in \{0,\ldots,9\}$.
- The data basis contains $n = 70'000$ images.

R Code for CNN Architecture on Spatial Data

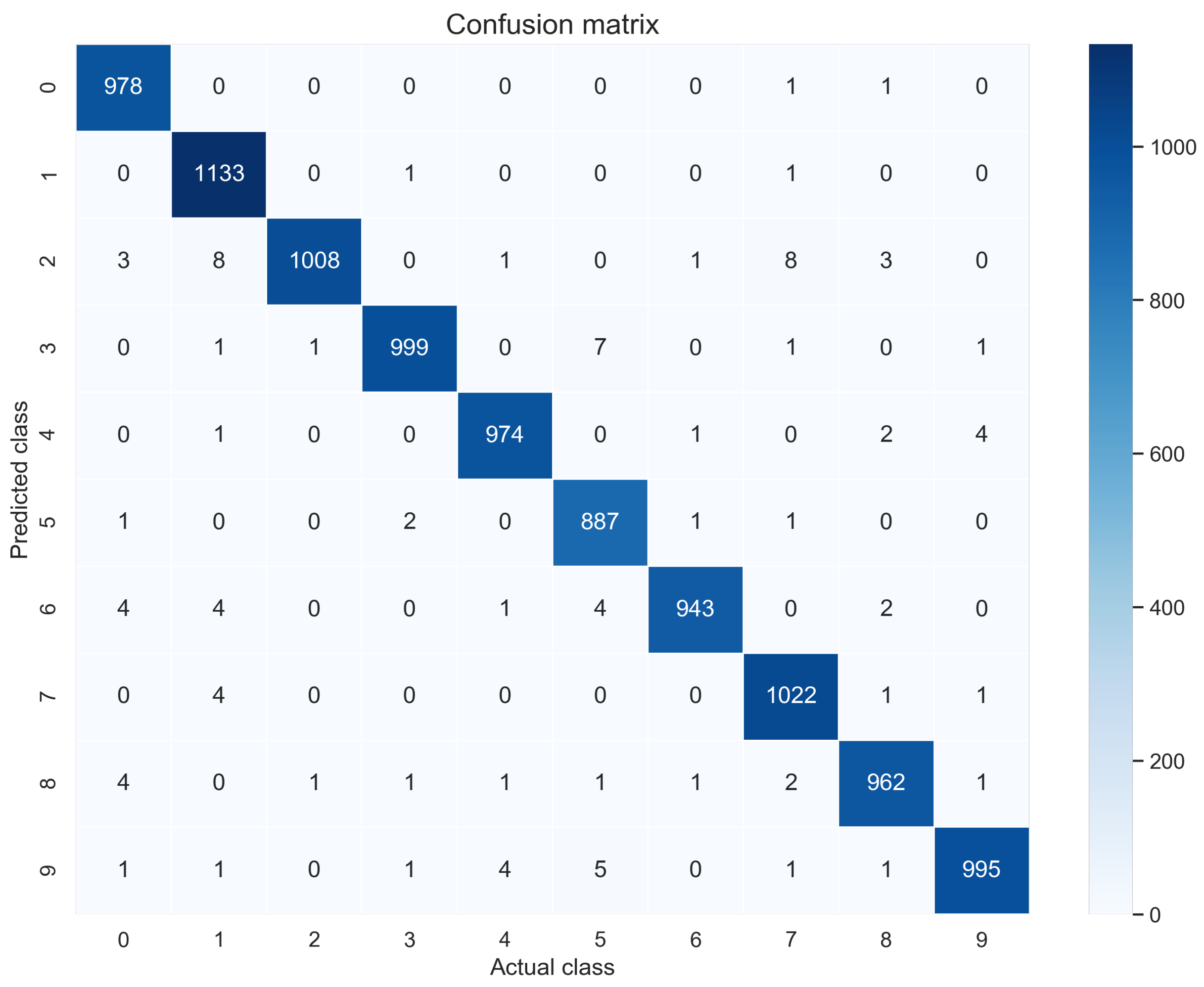
```r
model <- keras_model_sequential()

model %>%

layer_conv_2d(filters = 10, kernel_size = c(3,3), padding="valid",
                            activation = "linear", input_shape = c(28,28,1)) %>%
layer_batch_normalization() %>%
layer_activation(activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2,2), strides=c(2,2), padding="valid") %>%

layer_conv_2d(filters = 20, kernel_size = c(3,3), padding="valid") %>%
layer_batch_normalization() %>%
layer_activation(activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2,2), strides=c(1,1), padding="valid") %>%

layer_conv_2d(filters = 40, kernel_size = c(3,3), padding="valid") %>%
layer_batch_normalization() %>%
layer_activation(activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2,2), strides=c(2,2), padding="valid") %>%

layer_flatten() %>%
layer_dense(units=10, activation = 'softmax')
```

Explicit CNN Architecture and Network Parameters

```
Layer (type)                     Output Shape              Param #
==================================================================
conv2d_8 (Conv2D)                (None, 26, 26, 10)        100
------------------------------------------------------------------
batch_normalization_7 (BatchNormalizat (None, 26, 26, 10)  40
------------------------------------------------------------------
activation_7 (Activation)        (None, 26, 26, 10)        0
------------------------------------------------------------------
max_pooling2d_7 (MaxPooling2D)   (None, 13, 13, 10)        0
------------------------------------------------------------------
conv2d_9 (Conv2D)                (None, 11, 11, 20)        1820
------------------------------------------------------------------
batch_normalization_8 (BatchNormalizat (None, 11, 11, 20)  80
------------------------------------------------------------------
activation_8 (Activation)        (None, 11, 11, 20)        0
------------------------------------------------------------------
max_pooling2d_8 (MaxPooling2D)   (None, 10, 10, 20)        0
------------------------------------------------------------------
conv2d_10 (Conv2D)               (None, 8, 8, 40)          7240
------------------------------------------------------------------
batch_normalization_9 (BatchNormalizat (None, 8, 8, 40)    160
------------------------------------------------------------------
activation_9 (Activation)        (None, 8, 8, 40)          0
------------------------------------------------------------------
max_pooling2d_9 (MaxPooling2D)   (None, 4, 4, 40)          0
```

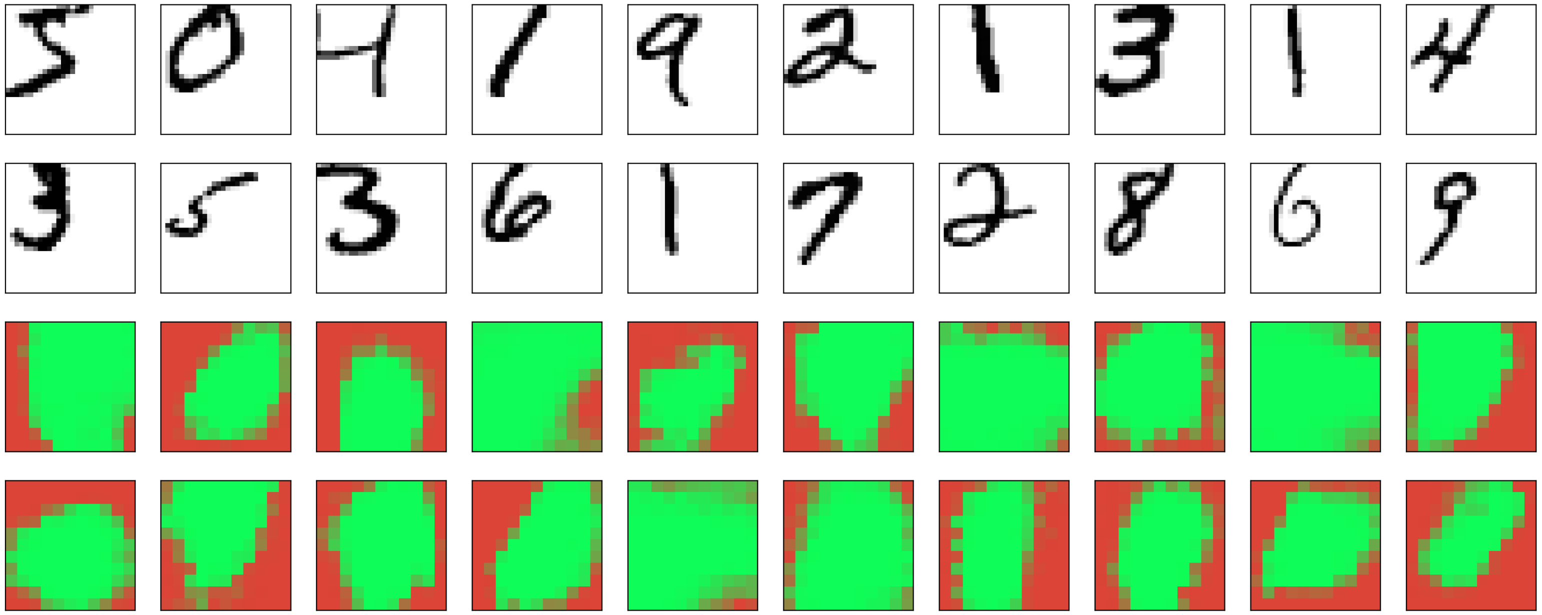
```
_________________________________________________________________________________________
flatten_3 (Flatten)                    (None, 640)                          0
_________________________________________________________________________________________
dense_3 (Dense)                        (None, 10)                           6410
=========================================================================================
Total params: 15,850
Trainable params: 15,710
Non-trainable params: 140 (1/2 of batch normalizations)
```

Result: Confusion Matrix

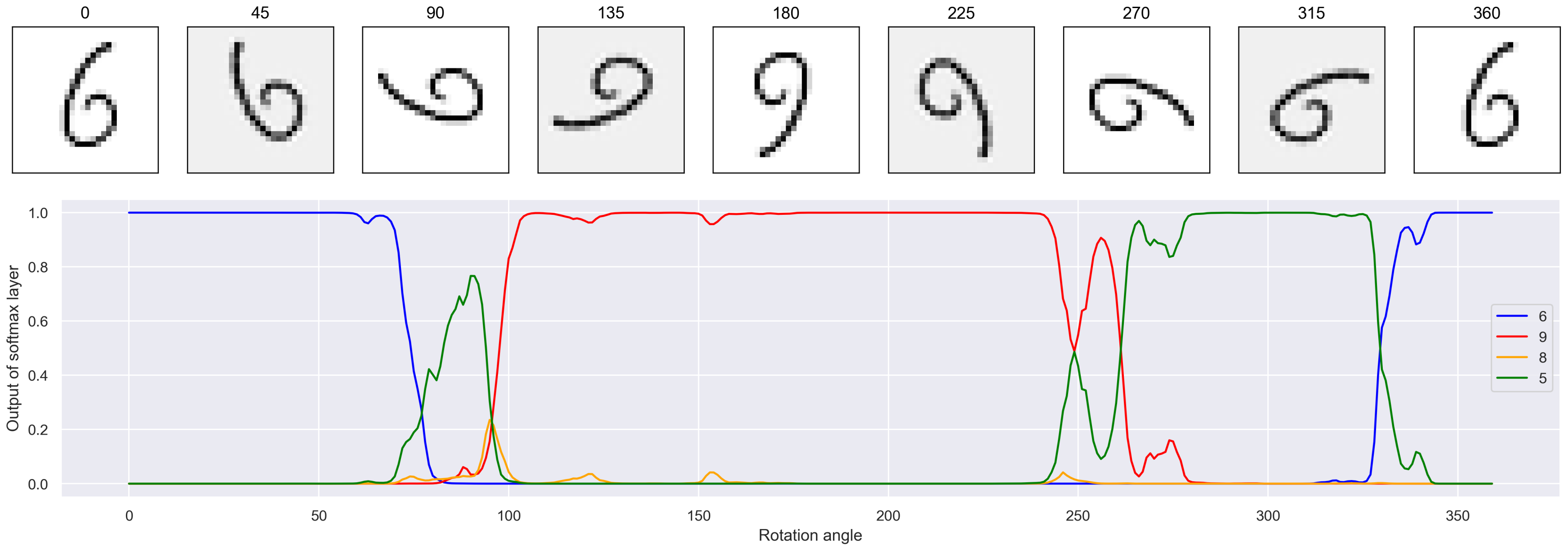


Predicted \ Actual	0	1	2	3	4	5	6	7	8	9
0	978	0	0	0	0	0	0	1	1	0
1	0	1133	0	1	0	0	0	1	0	0
2	3	8	1008	0	1	0	1	8	3	0
3	0	1	1	999	0	7	0	1	0	1
4	0	1	0	0	974	0	1	0	2	4
5	1	0	0	2	0	887	1	1	0	0
6	4	4	0	0	1	4	943	0	2	0
7	0	4	0	0	0	0	0	1022	1	1
8	4	0	1	1	1	1	1	2	962	1
9	1	1	0	1	4	5	0	1	1	995

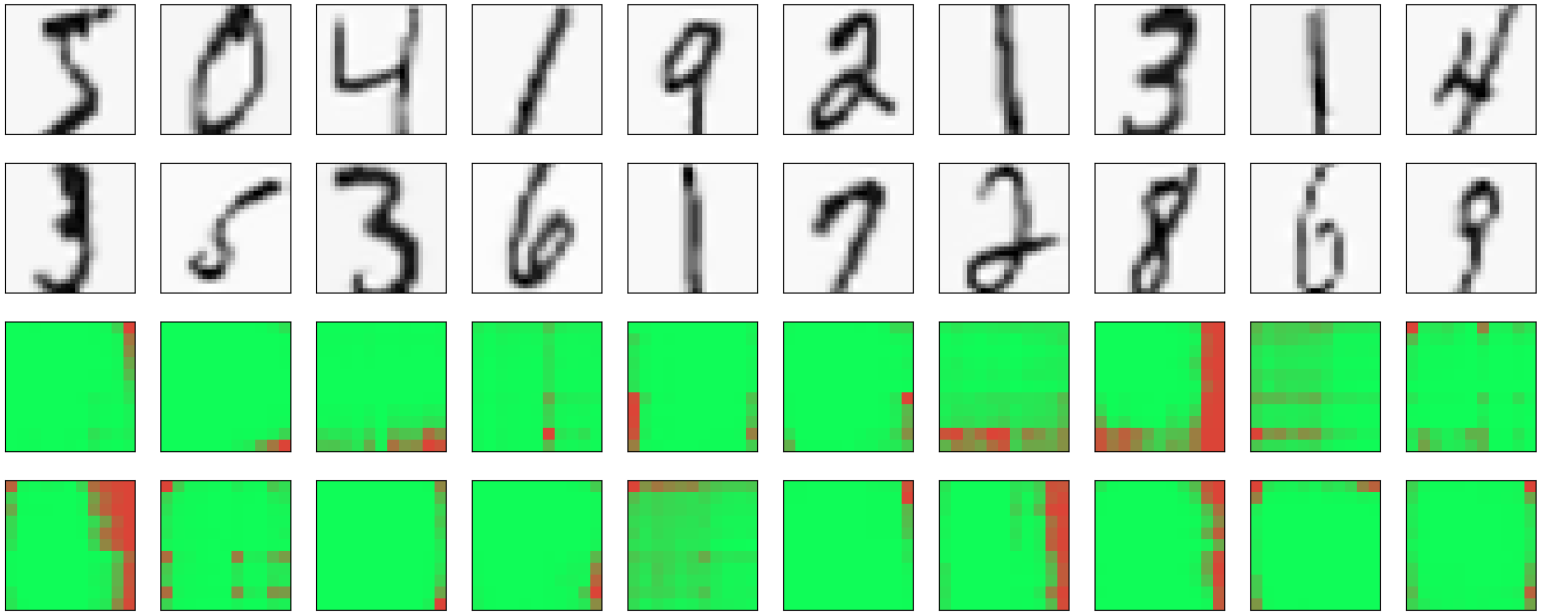
Shift Invariance



Rotation Invariance



Scale Invariance



References

- Efron, Hastie (2016). Computer Age Statistical Inference: Algorithms, Evidence, and Data Science. Cambridge UP.
- Gao, Wüthrich (2019). Convolutional neural network classification of telematics car driving data. Risks 7/1, article 6.
- Goodfellow, Bengio, Courville (2016). Deep Learning. MIT Press.
- Hastie, Tibshirani, Friedman (2009). The Elements of Statistical Learning. Springer.
- Meier, Wüthrich (2020). Convolutional neural network case studies: (1) anomalies in mortality rates (2) image recognition. SSRN 3656210.
- Perla, Richman, Scognamiglio, Wüthrich (2021). Time-series forecasting of mortality rates using deep learning. Scandinavian Actuarial Journal 2021/7, 572-598.
- Wiatowski, Bölcskei (2018). A mathematical theory of deep convolutional neural networks for feature extraction. IEEE Transactions on Information Theory 64/3, 1845-1866.
- Wüthrich, Merz (2021). Statistical Foundations of Actuarial Learning and its Applications. SSRN 3822407.