Machine Intelligence:: Deep Learning
Week 6

Beate Sick, Oliver Dürr, Jonas Brändli

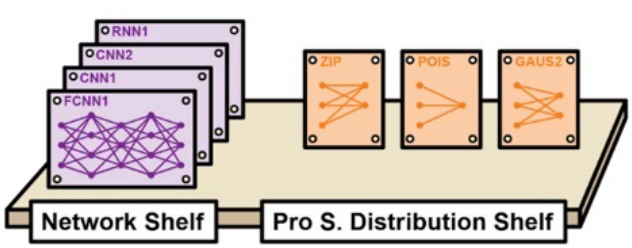
Probablistic models with flexible CPDs

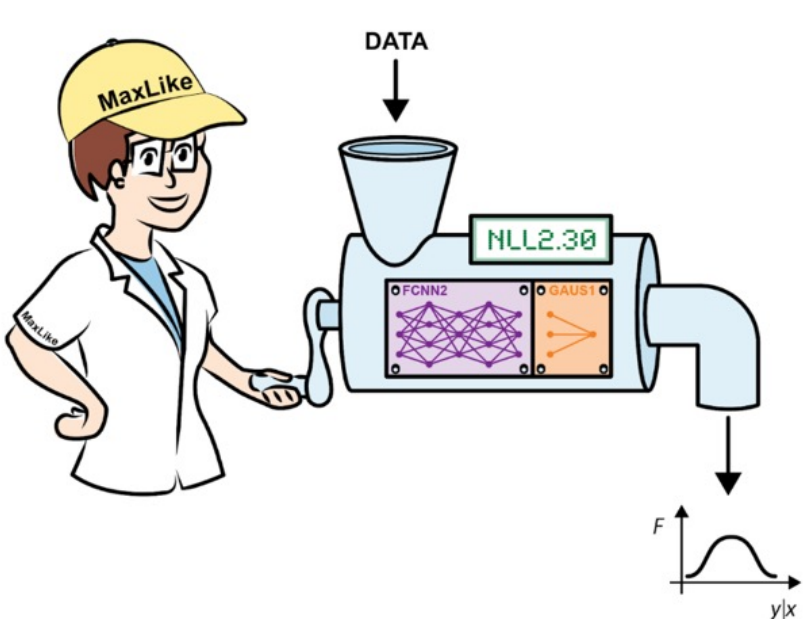
Outline of the DL Module (tentative)

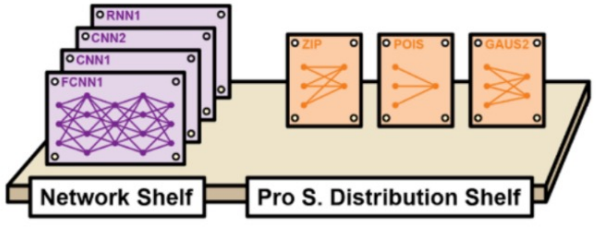
- Day 1: Jumpstart to DL
  - What is DL
  - Basic Building Blocks
  - Keras
- Day 2: CNN I
  - ImageData
- Day 3: CNN II and RNN
  - Tips and Tricks
  - Modern Architectures
  - 1-D Sequential Data
- Day 4: Looking at details
  - Linear Regression
  - Backpropagation
    - Resnet
  - Likelihood principle
- Day 5: Probabilistic Aspects
  - Likelihood principle (cont'd)
  - TensorFlow Probability (TFP)
  - Negative Loss Likelihood NLL
  - Count Data
- Day 6: Probabilistic models in the wild
  - Complex Distributions
  - Bayesian Modeling
- Day 7: Uncertainty in DL
  - Bayesian Modeling
- Day 8: Uncertainty cont'd
  - Bayesian Neural Networks
  - Projects

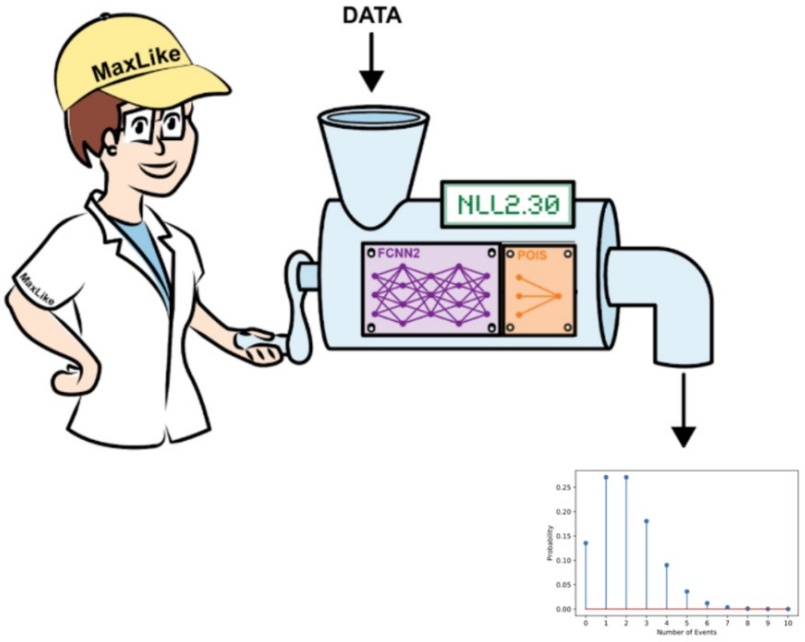
Projects please register (see website)
https://docs.google.com/spreadsheets/d/18VFrPbKq3YSOg8Ebc1q1wGgkfgaWl7IkcCCIGEDGj6Q/edit#gid=0

We have a flexible tool where the choice of the architecture and the choice of the outcome distribution is independent









Modeling count data continued

Recall the camper example

N=250 groups visiting a national park

Y=count: number of fishes cought

X1=persons: number of persons in group

X2=child: number of children in the group

X3=bait: indicates of life bait was used

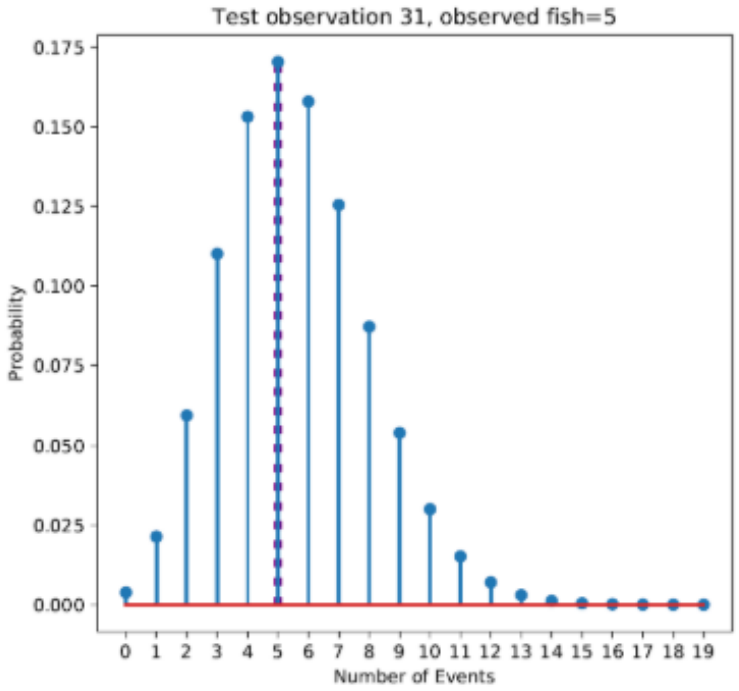
X4=camper: indicates if camper is brought

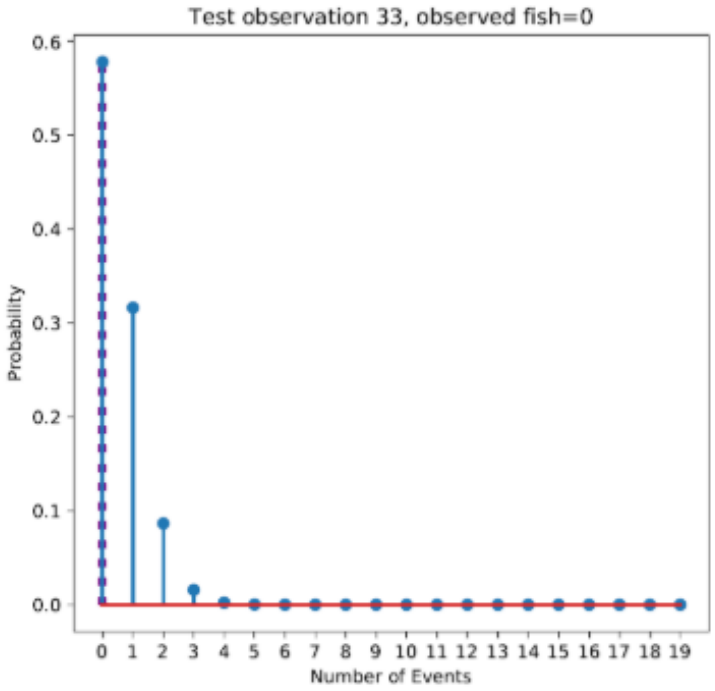
Data: https://stats.idre.ucla.edu/r/dae/zip

Model 2: Poisson regression, predicted CPDs for test observations

Predict CPD for outcome in test data:

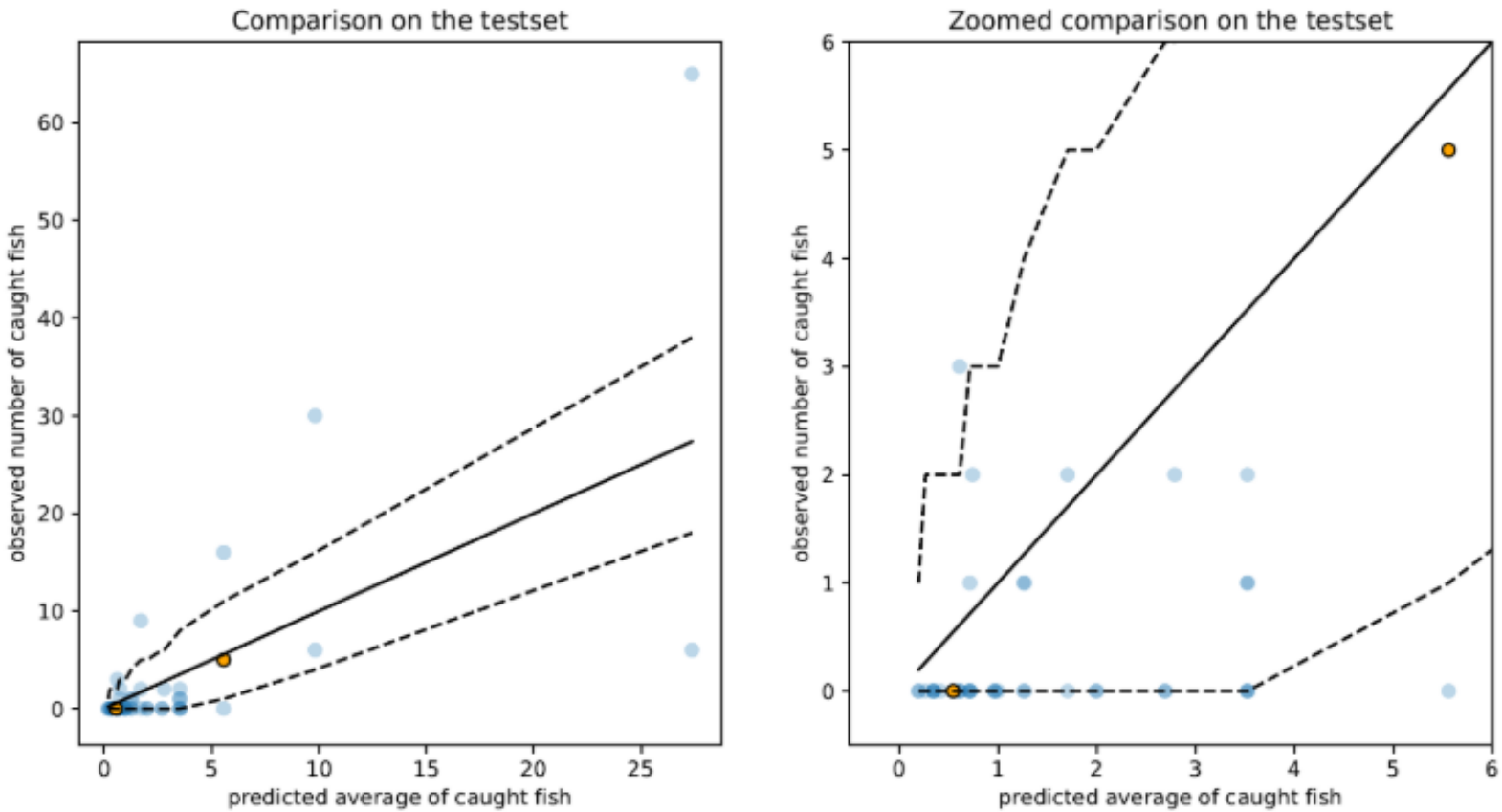
Group 31 used livebait, had a camper and were 4 persons with one child. Y=5 fish.
Group 33 used livebait, didn't have a camper and were 4 persons with two childern. Y=0 fish.





What is the likelihood of the observed outcome in test obs 31 and 33?

Model 2: Poisson regression, visualize the CPDs by quantiles



The mean of the CPD is depicted by the solid lines.
The dashed lines represent the 0.025 and 0.975 quantiles,
yielding the borders of a 95% prediction interval.

Note that different combinations of predictor values can yield the same parameters of the CPD.

Besprechung der Aufgabe

Use NN and tfp to fit a poisson model

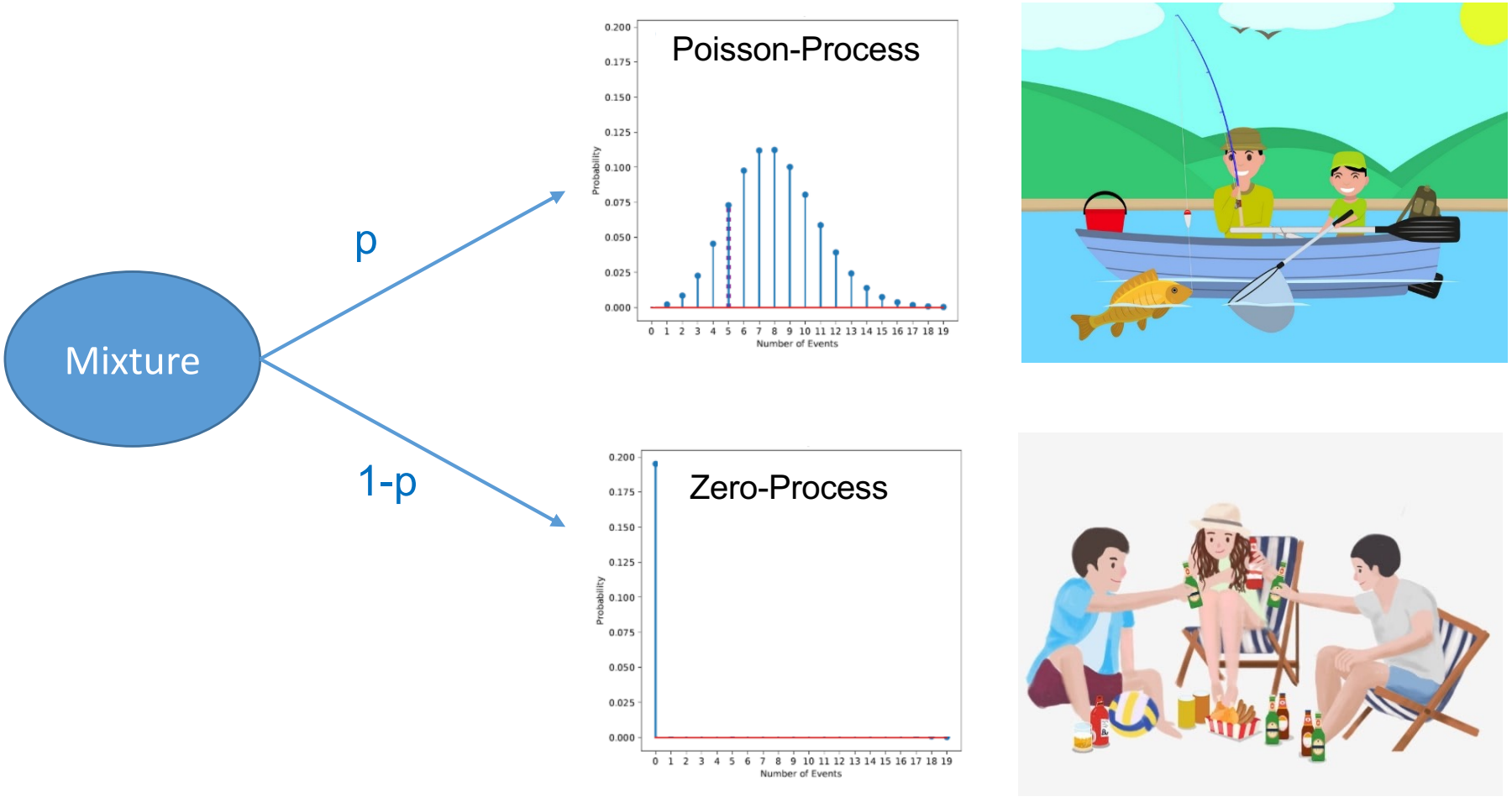
https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/14_poisreg_with_tfp.ipynb

Modeling count data:

M3: ZIP regression

Zero-Inflated Poisson (ZIP) as Mixture Process

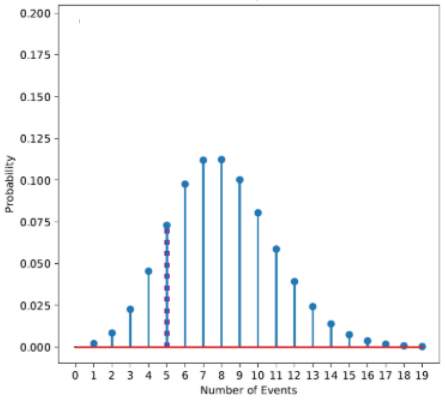
How many fish a group catches does not only depend on luck ;-)



Zero-Inflated Poisson (ZIP) can be seen as Mixture Distribution

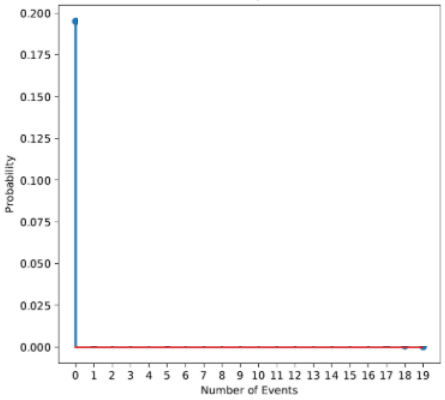
Poisson-Process

$p \cdot$



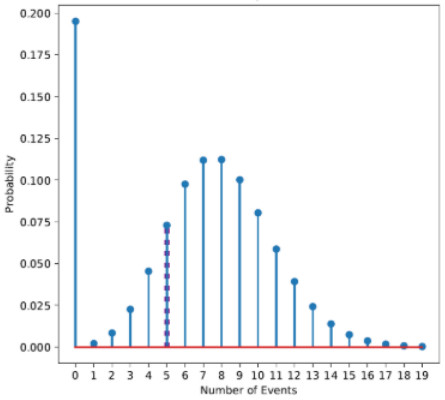
$+(1\text{-}p) \cdot$

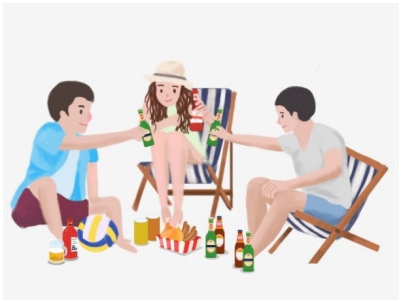
Zero-Process



$=$

Zero-inflated Poisson





Custom distribution for a ZIP distribution

```
def zero_inf(out):
    rate = tf.squeeze(tf.math.exp(out[:,0:1])) # First NN output controls lambda. Exp guarantee value >0
    s = tf.math.sigmoid(out[:,1:2]) # Second NN output controls p; sigmoid guarantees value in [0,1]
    probs = tf.concat([1-s, s], axis=1) # The two probabilities for 0's or Poissonian distribution
    return tfd.Mixture(
        cat=tfd.Categorical(probs=probs),# tfd.Categorical allows creating a mixture of two components
        components=[
        tfd.Deterministic(loc=tf.zeros_like(rate)), # Zero as a deterministic value
        tfd.Poisson(rate=rate), # Value drawn from a Poissonian distribution
    ])
```

https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/15_zipreg_with_tfp.ipynb

Model 3: Zero-Inflated Poisson regression via NNs in keras

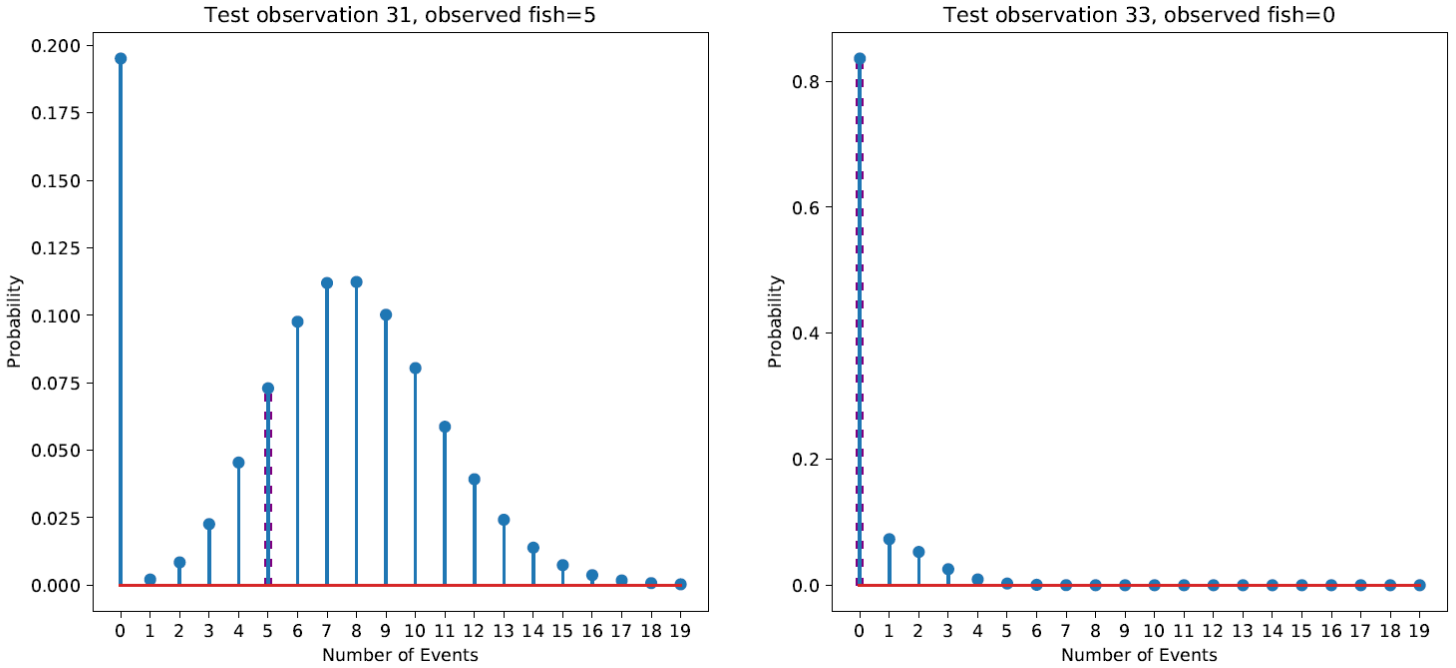
```
## Definition of the custom parameterized distribution

inputs = tf.keras.layers.Input(shape=(X_train.shape[1],))

out = Dense(2)(inputs) #A

p_y_zi = tfp.layers.DistributionLambda(zero_inf)(out)

model_zi = Model(inputs=inputs, outputs=p_y_zi)
```

https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/15_zipreg_with_tfp.ipynb

Model 3: ZIP regression, get test NLL from Gaussian CPD

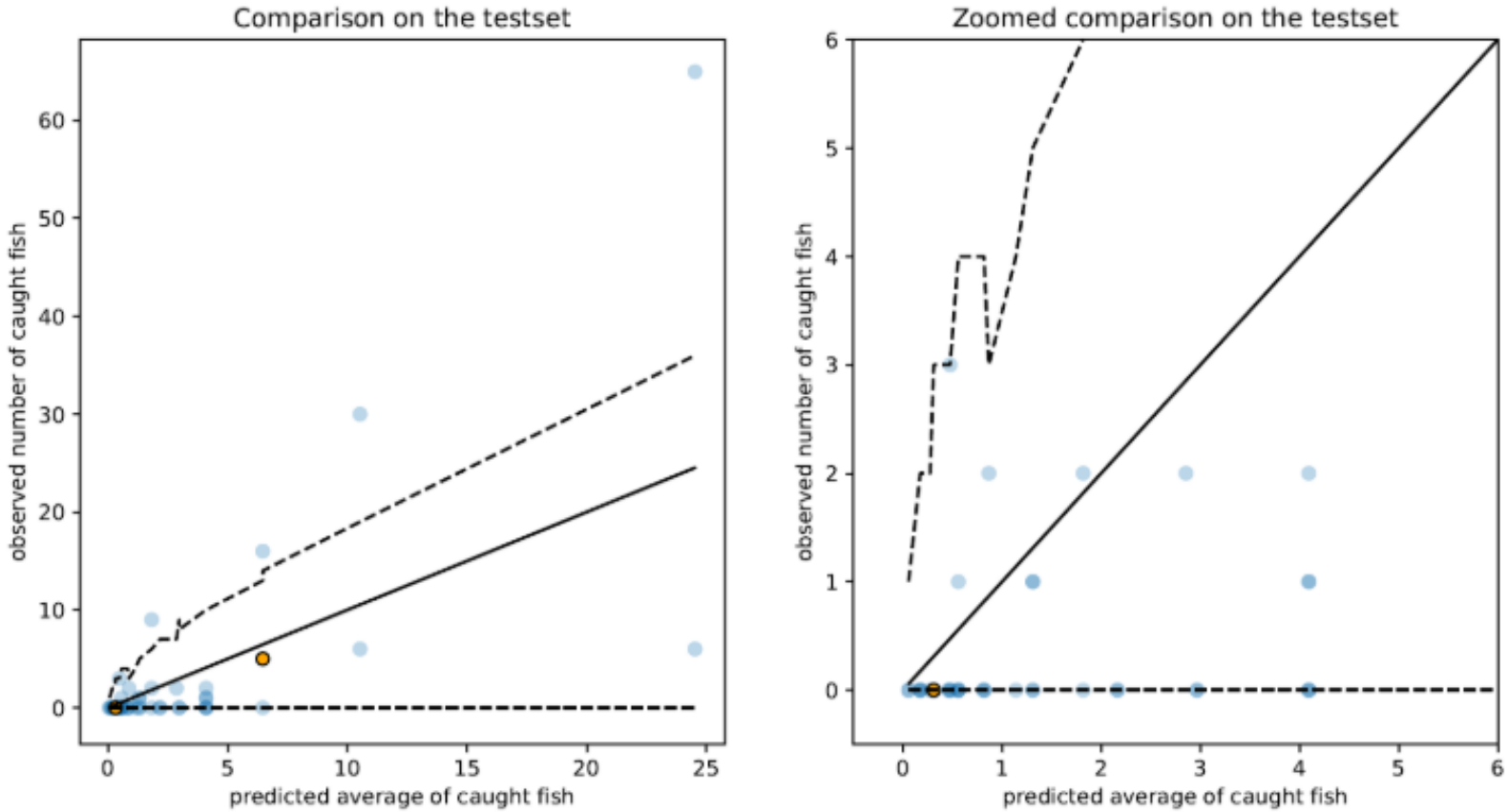
Predict CPD for outcome in test data:

Group 31 used livebait, had a camper and were 4 persons with one child. Y=5 fish.
Group 33 used livebait, didn't have a camper and were 4 persons with two childern. Y=0 fish.



What is the likelihood of the observed outcome in test obs 31 and 33?

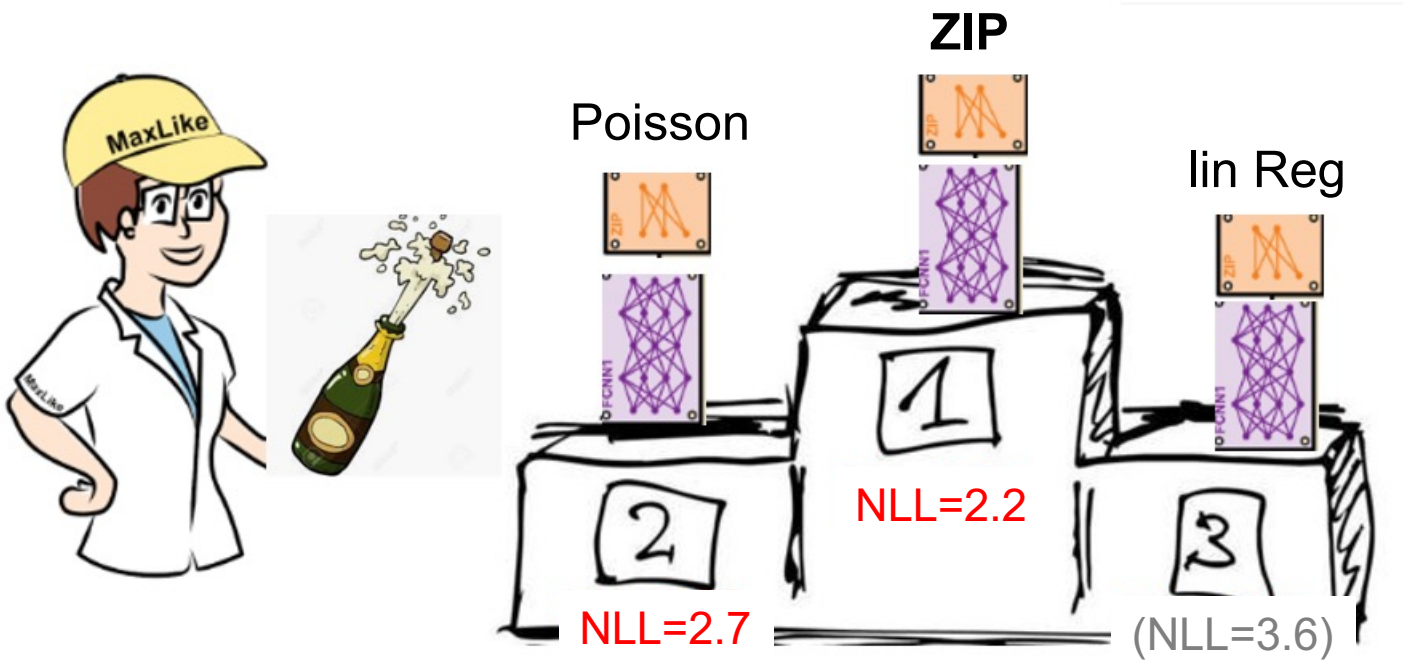
Model 3: ZIP regression, visualize the CPDs by quantiles



The mean of the CPD is depicted by the solid lines.
The dashed lines represent the 0.025 and 0.975 quantiles, yielding the borders of a 95% prediction interval.

Note that different combinations of predictor values can yield the same parameters of the CPD.

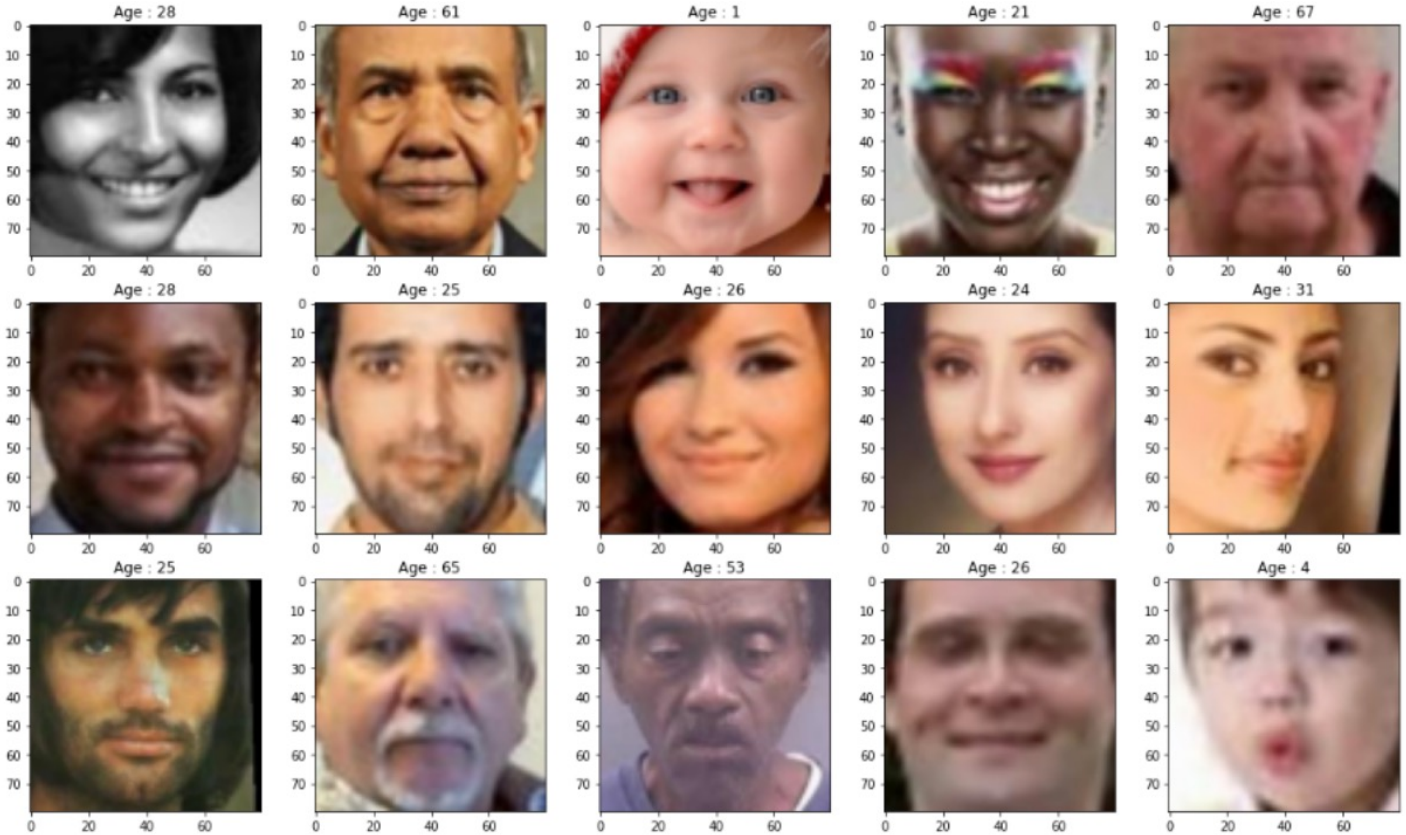
Validation NLL allows to rank different probabilistic models



We cant compare NLLs from discrete and continuous models

Probabilistic models with complex input data

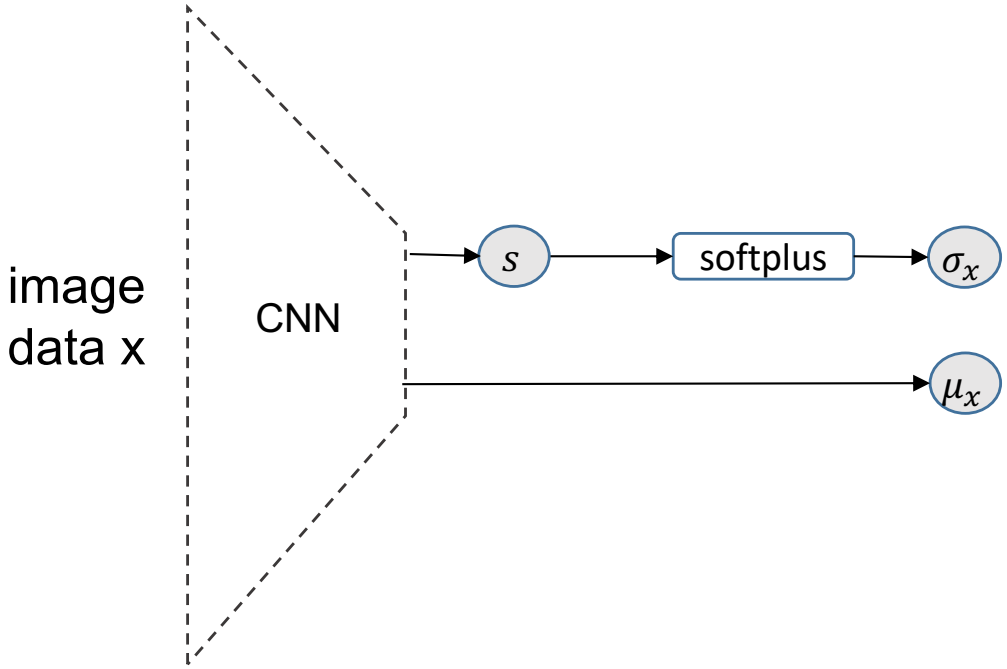
The UTK face data – face image data with known age



Data: https://stats.idre.ucla.edu/r/dae/zip

UTKFace data set containing N= 23'708 images of cropped faces of humans with known age ranging from 1 to 116 years.

Modeling a Gaussian CPD with flexible mean & variance



Minimize the negative log-likelihood (NLL):

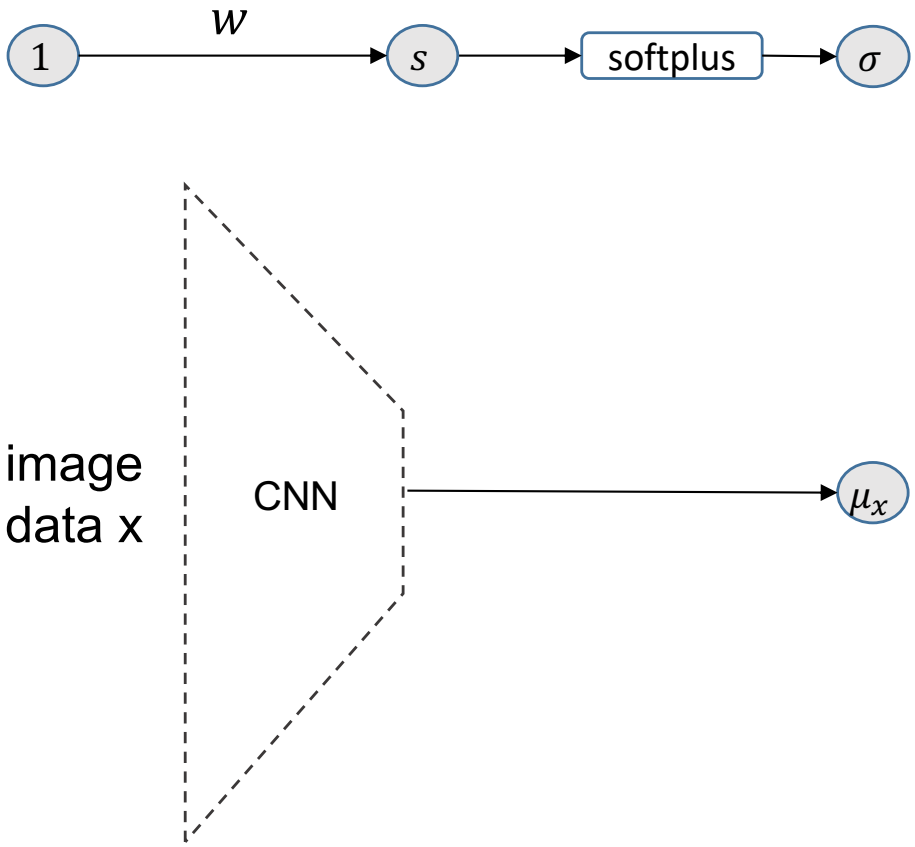
$$\hat{\mathbf{w}}_{\mathrm{ML}} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma^{2}}}\right) + \frac{\left(y_i - \mu_{x_i}\right)^2}{2\sigma^2}$$

CNNs for modeling Gaussian CPDs

```
def NLL(y, distr):
  return -distr.log_prob(y)

def my_dist(params):
  return tfd.Normal(loc=params[:,0:1], scale=1e-3 + tf.math.softplus(0.05 * params[:,1:2]))# both paramete
rs are learnable

inputs = Input(shape=(80,80,3))
x = Convolution2D(16,kernel_size,padding='same',activation="relu")(inputs)
x = Convolution2D(16,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Flatten()(x)
x = Dense(500,activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(50,activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(2)(x)
dist = tfp.layers.DistributionLambda(my_dist)(x)

model_flex = Model(inputs=inputs, outputs=dist)
model_flex.compile(tf.keras.optimizers.Adam(), loss=NLL)
```

We control both parameters (μ_x, σ_x) of a Gaussian CPD $N(\mu_x, \sigma_x)$ by a CNN
→ More flexible than in classical regression where $\sigma = constant$

Modeling Gaussian CPD with flexible mean & constant variance



Minimize the negative log-likelihood (NLL):

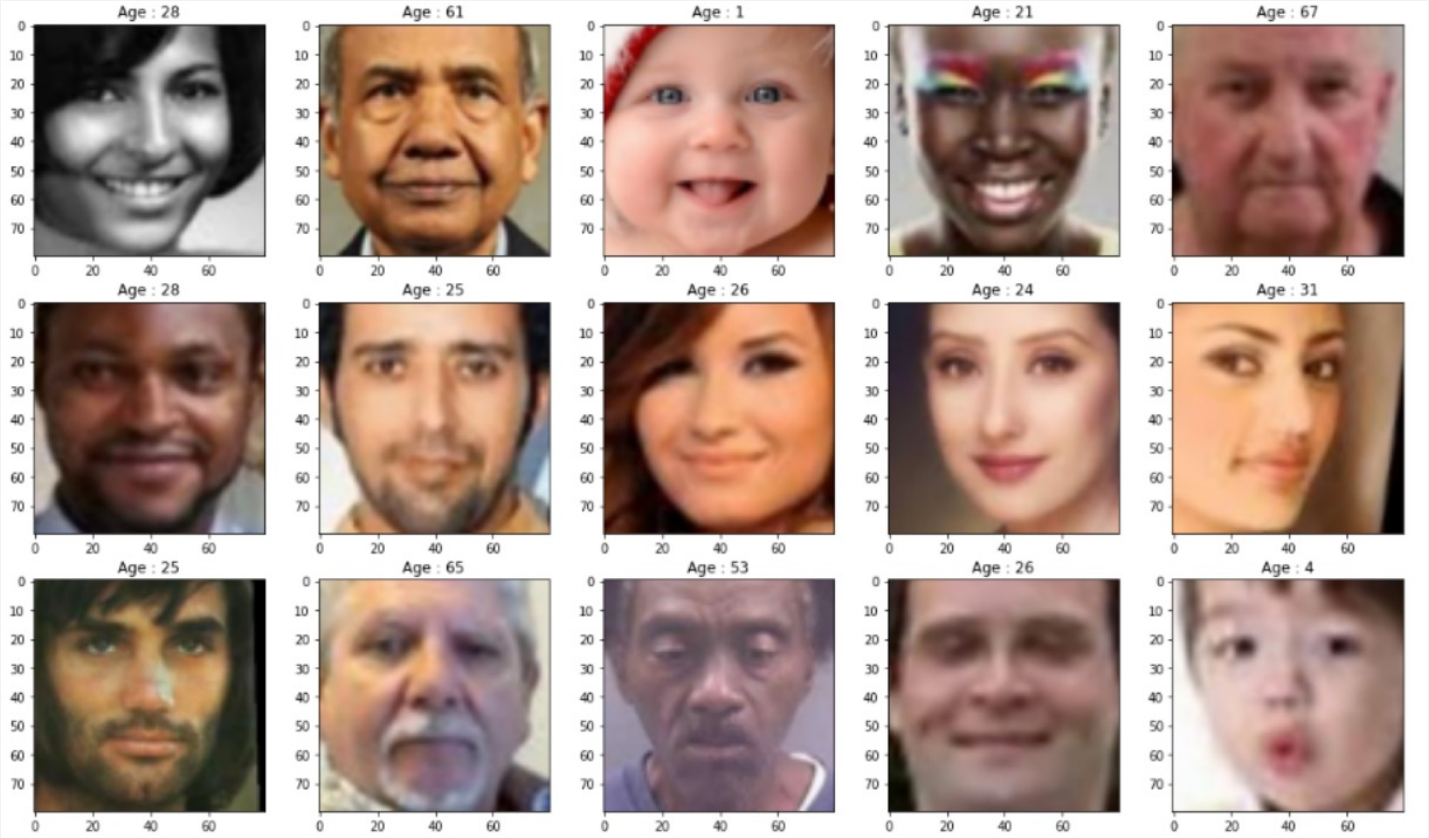
$$\hat{\mathbf{w}}_{\mathrm{ML}} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma^{2}}}\right) + \frac{\left(y_i - \mu_{x_i}\right)^2}{2\sigma^2}$$

CNNs for modeling Gaussian CPDs

```
def NLL(y, distr):
  return -distr.log_prob(y)

def my_dist(params):
  return tfd.Normal(loc=params[:,0:1], scale=1e-3 + tf.math.softplus(0.05 * params[:,1:2]))# both paramete
rs are learnable

input1 = Input(shape=(80,80,3))
input2 = Input(shape=(1,))
x = Convolution2D(16,kernel_size,padding='same',activation="relu")(input1)
x = Convolution2D(16,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Flatten()(x)
x = Dense(500,activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(50,activation="relu")(x)
x = Dropout(0.3)(x)
out1 = Dense(1)(x)
out2 = Dense(1)(input2)
params = Concatenate()([out1,out2])
dist = tfp.layers.DistributionLambda(my_dist)(params) #

model_const_sd = Model(inputs=[input1,input2], outputs=dist) ## use a trick with two inputs, input2 is jus
t ones
model_const_sd.compile(tf.keras.optimizers.Adam(), loss=NLL)
```

We control both parameters (μ, σ) of a Gaussian CPD $\mathrm{N}(\mu_x, \sigma)$ → But assume a constant variance

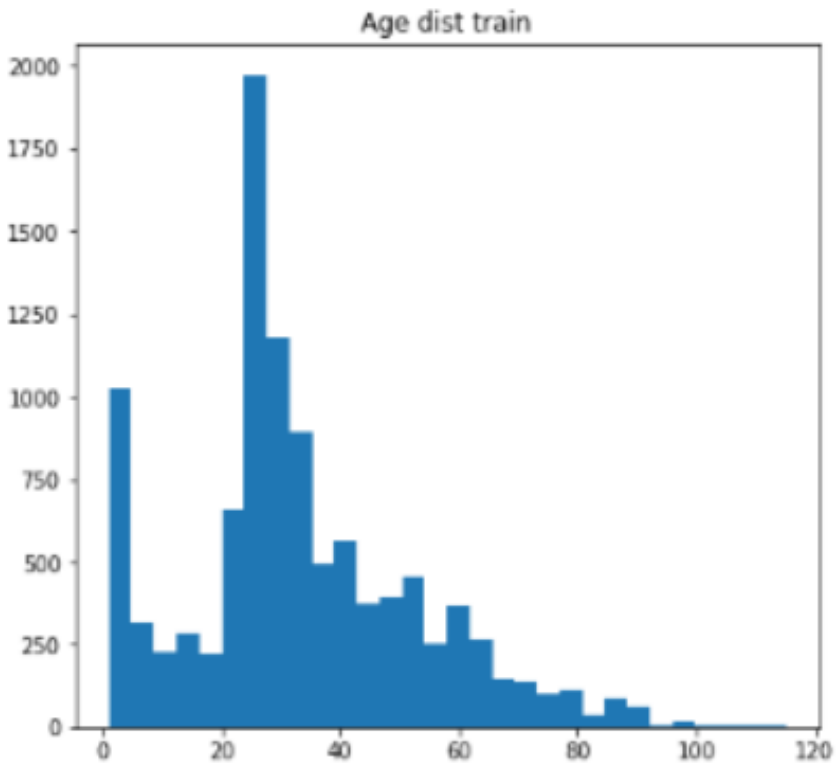
Excercise

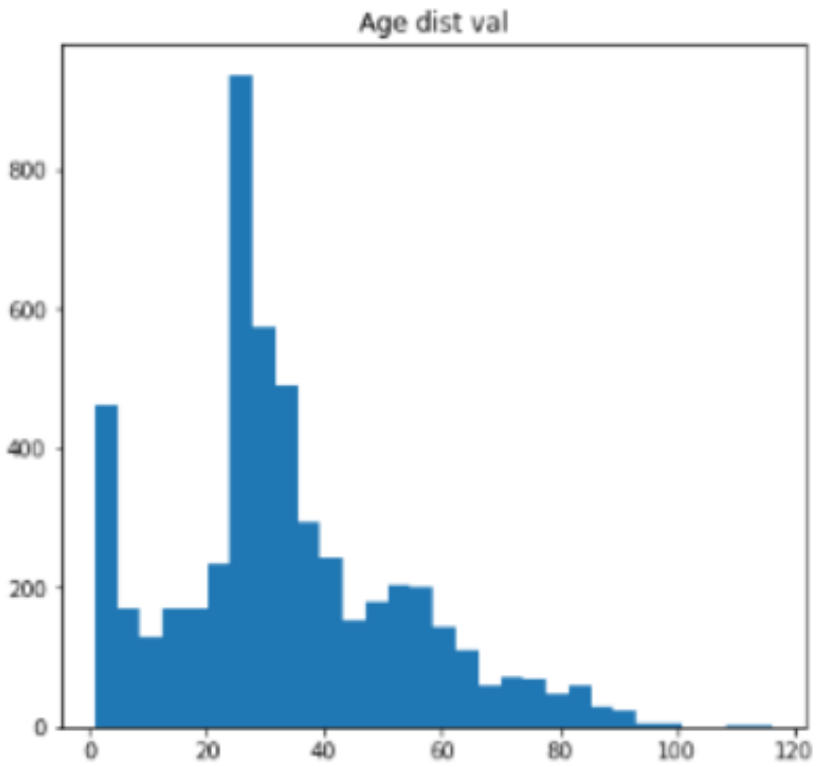


Check out the models for age prediction with flexible and constant variance and try to understand the code and to answer the questions.

https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/17_faces_regression.ipynb

Age distribution

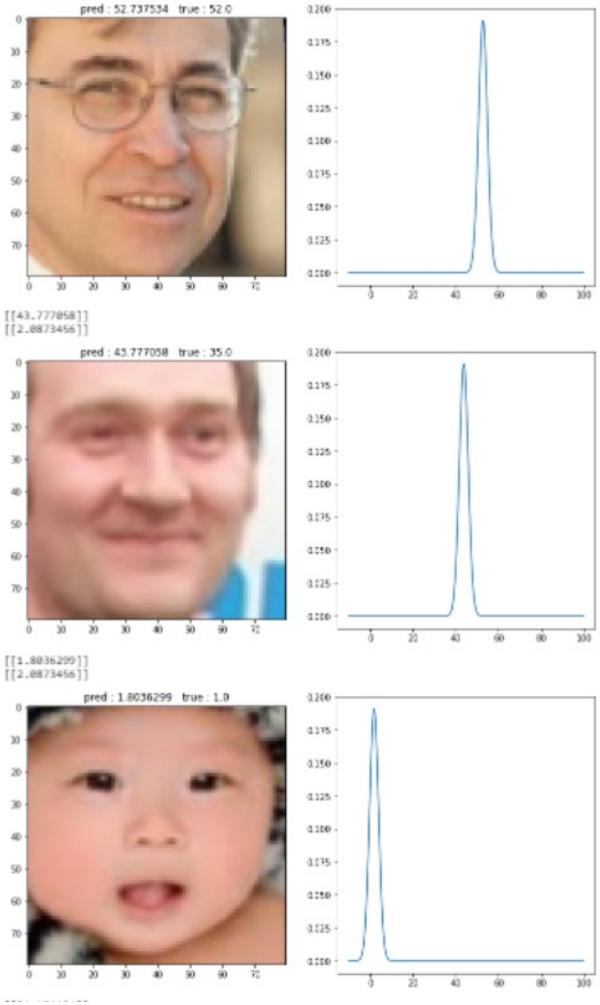




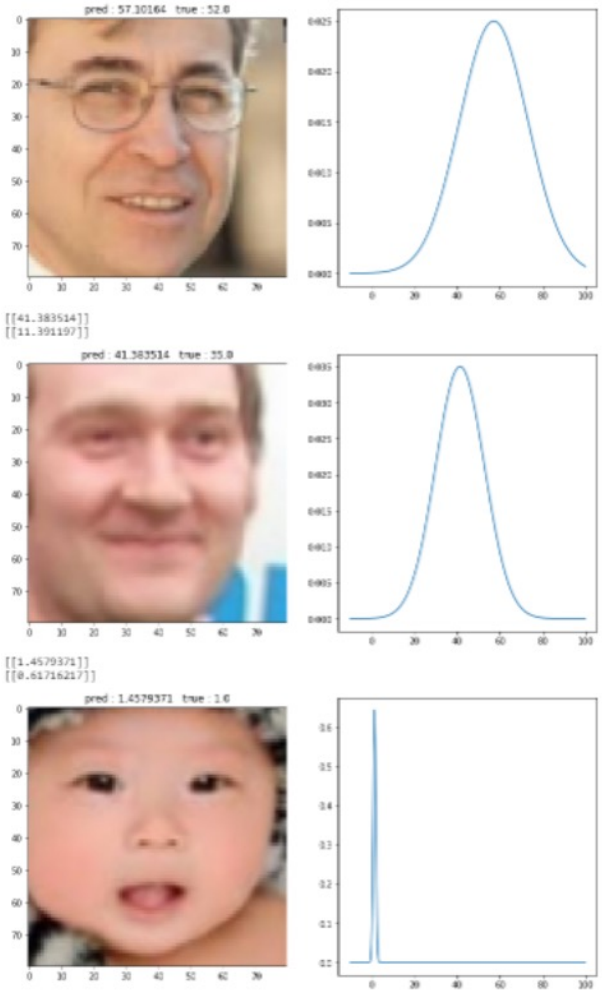
We have a lot of small children in the data set, for whom the age estimation is probably not so difficult.

Resulting age CPDs

Constant variance



flexible variance



In case of a flexible variance, a broad predicted Gaussian CPD does indicate high uncertainty about the age.

The MNIST data set – images of handwritten digits

The MNIST data set containing N= 60’000 images of handwritten digits of the ten classes 0, 1, 2, 3, 4, 5, 6,7, 8, 9,

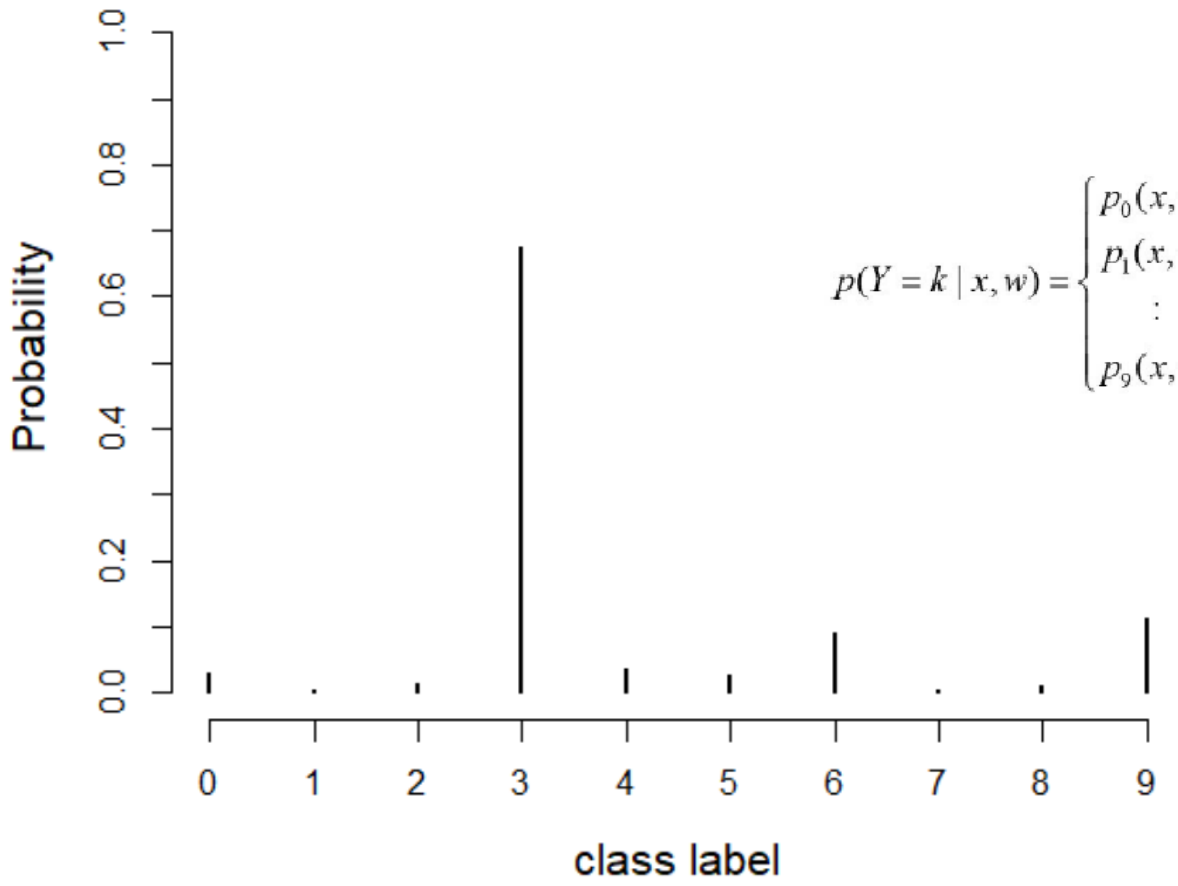
CNNs for modeling Gaussian CPDs

```
# here we define hyperparameter of the CNN
batch_size = 128
nb_classes = 10
img_rows, img_cols = 28, 28
kernel_size = (3, 3)
input_shape = (img_rows, img_cols, 1)
pool_size = (2, 2)
```

```
# define CNN with 2 convolution blocks and 2 fully connected layers
model = Sequential()

model.add(Convolution2D(8,kernel_size,padding='same',input_shape=input_shape))
model.add(Activation('relu'))
model.add(Convolution2D(8, kernel_size,padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))

model.add(Convolution2D(16, kernel_size,padding='same'))
model.add(Activation('relu'))
model.add(Convolution2D(16,kernel_size,padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))

model.add(Flatten())
model.add(Dense(40))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

# compile model and intitialize weights
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

We model 10 parameters $(p_0, p_1, \ldots p_9)$ of a Multinomial CPD (it could also be done with 9 parameters)
→ Most flexible CPD for an outcome with 10 possible values

We predict for each image a multinomial CPD

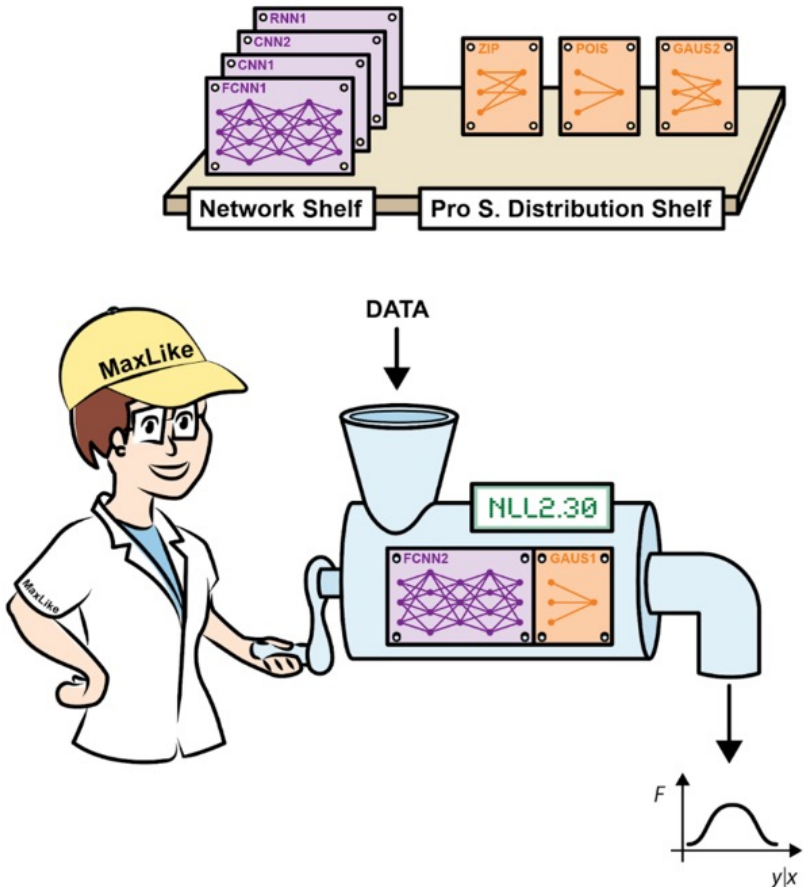


$$p(Y=k \mid x,w)=\begin{cases} p_0(x,w) & \text{for k=0} \\ p_1(x,w) & \text{for k=1} \\ \vdots & \vdots \\ p_9(x,w) & \text{for k=9} \end{cases} \quad \text{with} \sum_{i=0}^{9} p_i(x,w)=1$$

The multinomial distribution is especially flexible because it has as many parameters as possible values (or actually one parameter less, because probabilities need to sum up to one).

Count Distributions in the real world

Building state of the art networks



Two Design choices

1. Choose the right network architecture to exploit structure of your problem ("inductive bias")
   - Images CNN
   - Sequences RNN or 1D convolution

2. Choose the right CPD for your data

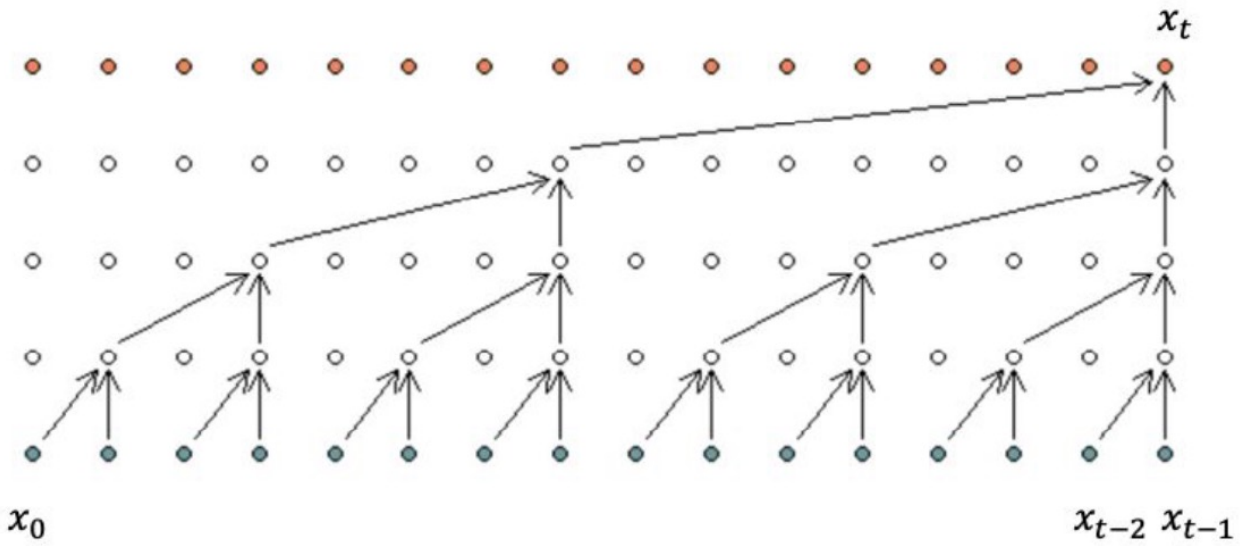
- Train with NLL
- Evaluate performance on validation with NLL

Two state-of-the art models PixelCNN and Wavenet have been improved by using *discretized logistic mixture* distributions as CPD.

PixelCNN and Wavenet In/Output

- Both are autoregressive models $p(x_t | x_{t-1}, \dots x_0)$
  - Wavenet
    - Predicts the next audio sample x_t from all previous $x_{t-1}, \dots x_0$
    - x_t can take a number between 0 and 2^16-1 = 65,535
  - PixelCNN
    - Predicts the next pixel x_t from all previous $x_{t-1}, \dots x_0$
    - x_t can take a value between 0 and 2^8-1 = 255

Wavenet architecture (see lecture 06_sequences)



PixelCNN and Wavenet In/Output

Modelling $p(x_t | x_{t-1}, \ldots x_0)$

- Naive approach output probabilities for the different classes
  - This has been done in first versions
  - Does not take nature of the output data into account

- Better find a *flexible* distribution to model the discrete data between 0 and say 255
  - Both architectures have been improved by mixtures of discrete logistics

We now have a look at the mixtures of discrete logistics.

The code for the figures is in:
https://github.com/tensorchiefs/dl_book/blob/master/chapter_06/nb_ch06_01.ipynb

Making sense of discretized logistic mixture

- The base distribution: the logistic with 2 parameters
  - loc kind of center
  - scale spread

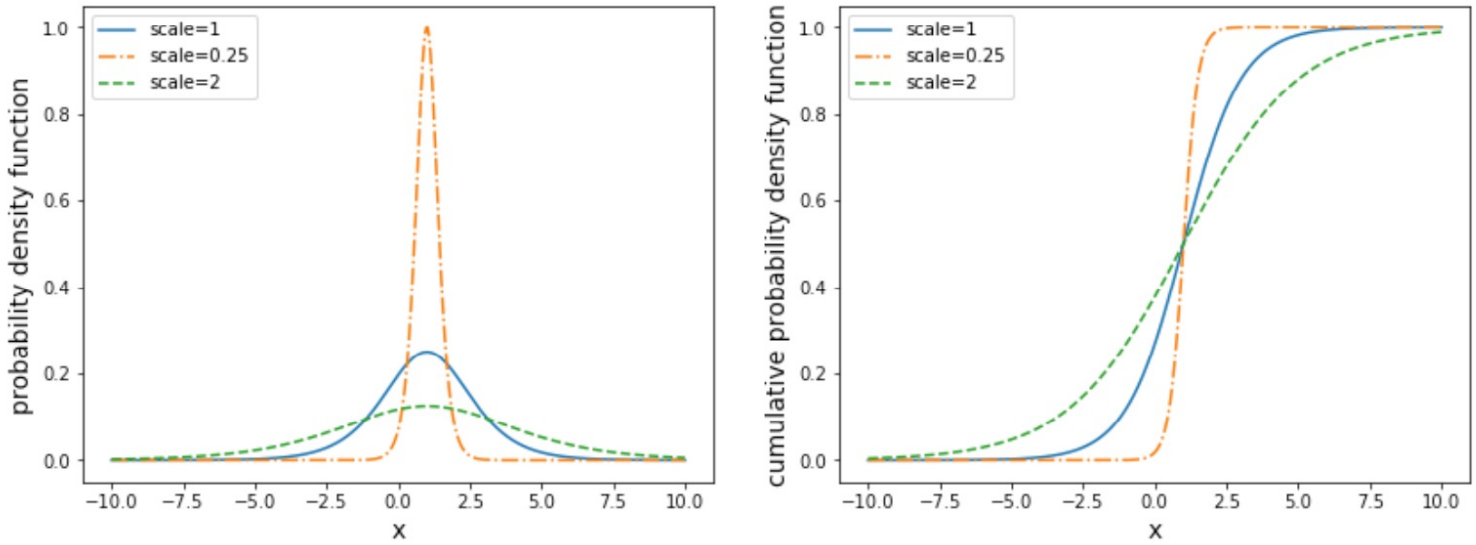
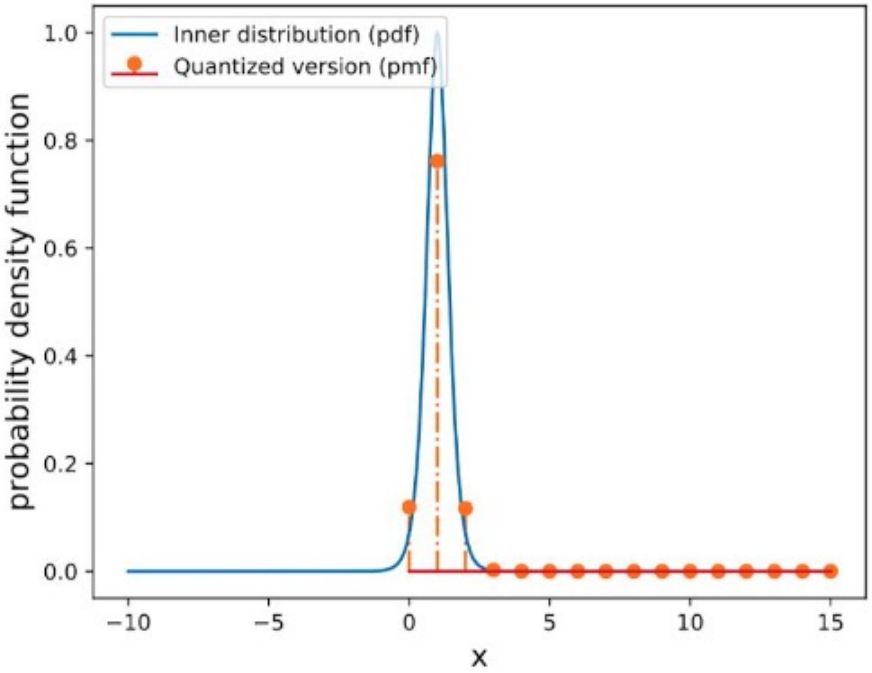


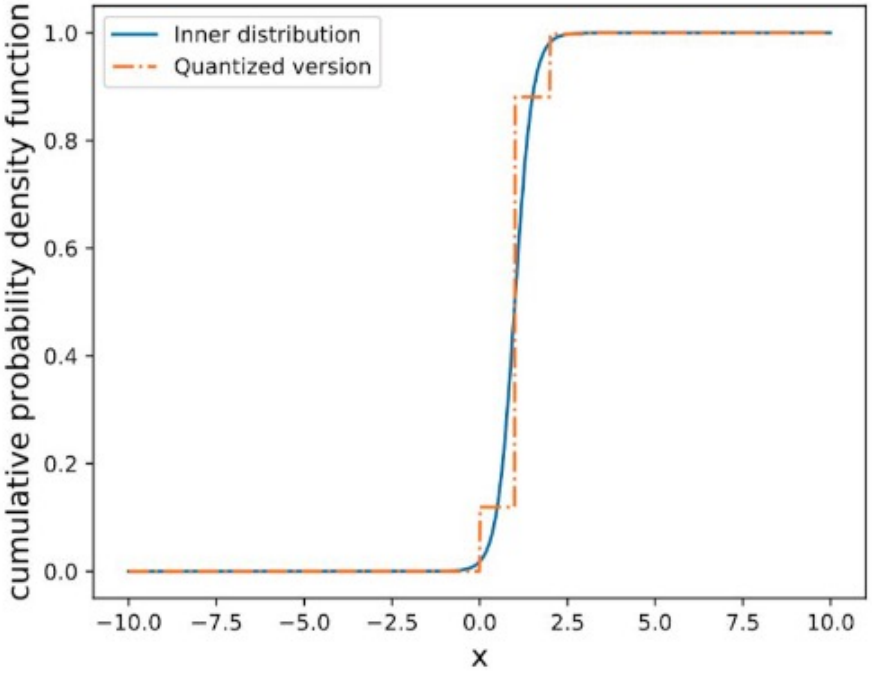
Figure 6.3 Three logistic functions created using tfd.Logistic(loc=1, scale=scale) with values of 0.5, 1.0, and 2.0 for the scale parameter. On the left is the probability density function (PDF) and on the right, the cumulative probability density function (CDF).

- The CFD has the same functional form as the sigmoid activation
- Still not discrete

Making sense of discretized logistic mixture

- The quantized (discretized) base distribution the logistic





Making sense of discretized logistic mixture

- Mixing several (here 2) discretized logistics

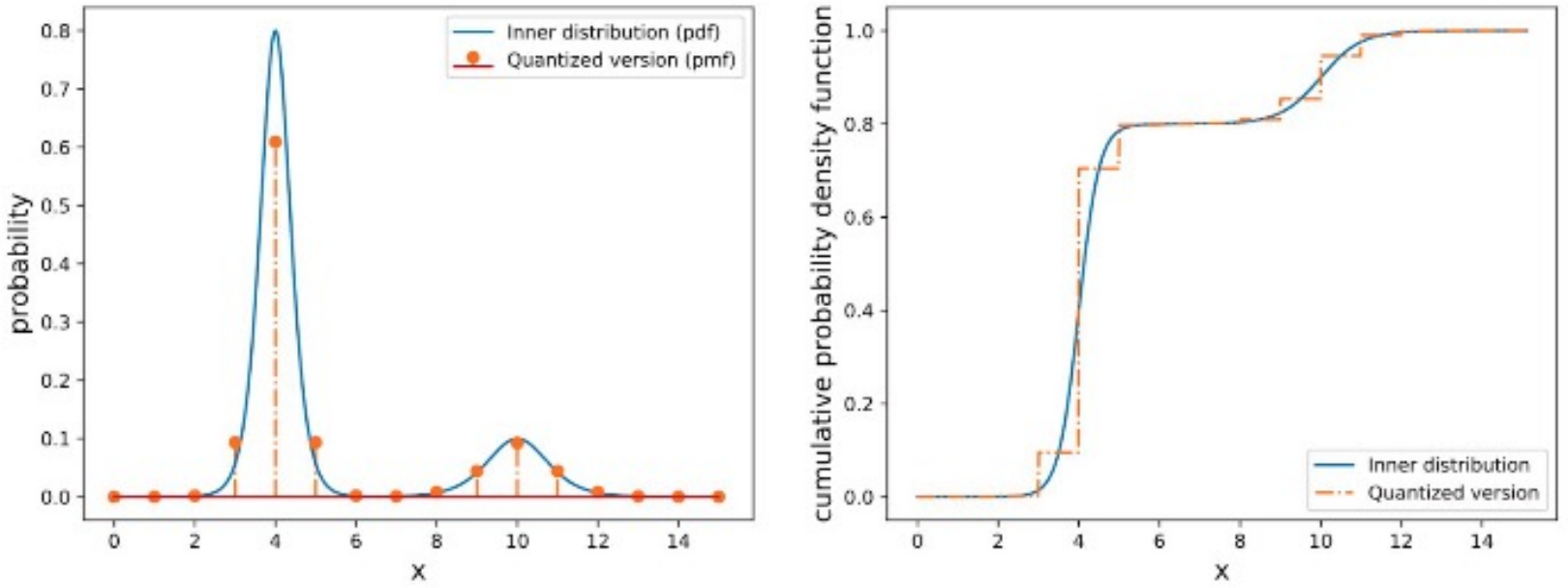


Figure 6.5 The resulting discrete distribution when mixing two logistic distributions (see listing 6.2 for the code that produces these plots)

Number of components, which need to be determined by network?

- 2 (or 1) for mixing
- 2 for scale
- 2 for location

Case Study for discretized logistic mixture

The original PixelCNN++ and Wavenet take too long to train. The fish data is too small for deep learning models. Let's use a large count data set.

Goal: Probabilistic model for deer activity conditioned on the time (in day and year).

wild	year	time	daytime	weekday
0	2002.0	0.000000	night.am	Sunday
0	2002.0	0.020833	night.am	Sunday
...	....	....	....	...
1	2002.0	0.208333	night.am	Sunday
0	2002.0	0.229167	pre.sunrise.am	Sunday
0	2002.0	0.270833	pre.sunrise.am	Sunday

The columns have the following meaning:

wild: the number deers killed in a road accident in Bavaria

year: the year (from 2002 to 2009 in the training set from 2010 to 2011 for the test set)

time: the number of days to the first event. These numbers are measured in fractions of a day.

Data on deer related car accidents in the years 2002 until 2011 in Bavaria, Germany.
Target variable (wild): use number of deers killed during 30 minute period as surrogate

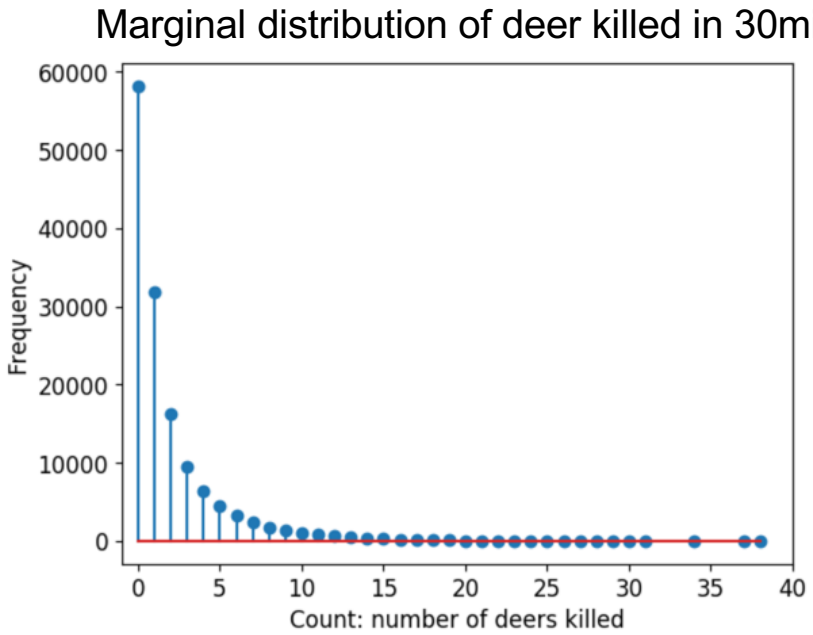
Modeling count data

Goal:
Predict CPD for y=#deers-killed-in-30min, given x (time and derived variables).
Possible CPD models:

$$(y \mid x) \sim \text{Pois}(\lambda_x)$$

$$(y \mid x) \sim \text{ZIP}({}^{z}p_x, \lambda_x)$$

$$(y \mid x) \sim \text{discretizedLogisticMix}({}^{1}p_x, {}^{2}p_x\ {}^{3}p_x, {}^{1}\mu_x, {}^{2}\mu_x, {}^{3}\mu_x, {}^{1}\sigma_x, {}^{2}\sigma_x, {}^{3}\sigma_x)$$



Code

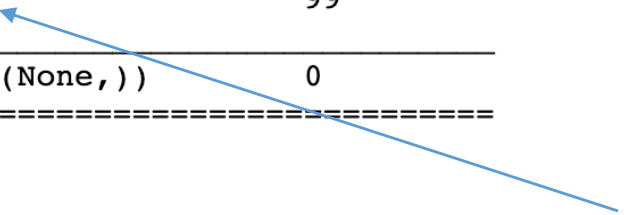
```
def quant_mixture_logistic(out, bits=8, num=3):
    loc, un_scale, logits = tf.split(out,#A
                                     num_or_size_splits=num,
                                     axis=-1)
    scale = tf.nn.softplus(un_scale) #B
    discretized_logistic_dist = tfd.QuantizedDistribution(
      distribution=tfd.TransformedDistribution( #C
          distribution=tfd.Logistic(loc=loc, scale=scale),
          bijector=tfb.AffineScalar(shift=-0.5)),
      low=0.,
      high=2**bits - 1.
    )
    mixture_dist = tfd.MixtureSameFamily(#D
        mixture_distribution=tfd.Categorical(logits=logits),
        components_distribution=discretized_logistic_dist)
    return mixture_dist

```

```
inputs = tf.keras.layers.Input(shape=(100,))
h1 = Dense(10, activation='tanh')(inputs)
out = Dense(6)(h1) #E
p_y = tfp.layers.DistributionLambda(quant_mixture_logistic)(out)
```

https://github.com/tensorchiefs/dl_book/blob/master/chapter_06/nb_ch06_01.ipynb

Architectures

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 16)]              0
_________________________________________________________________
dense_10 (Dense)             (None, 100)               1700
_________________________________________________________________
dense_11 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1010
_________________________________________________________________
dense_13 (Dense)             (None, 9)                 99
_________________________________________________________________
distribution_lambda_4 (Distr ((None,), (None,))        0
=================================================================
Total params: 12,909
Trainable params: 12,909
Non-trainable params: 0
_________________________________________________________________
```



Poisson

(None, 1)

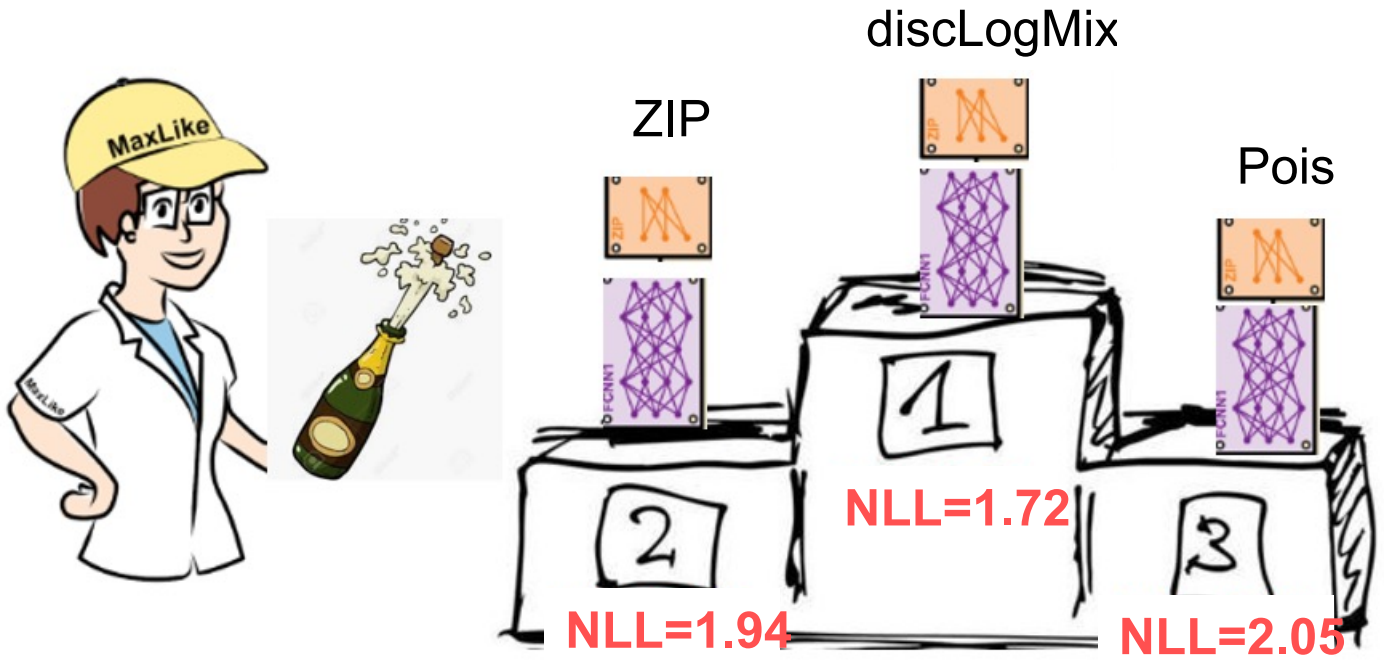
Zero Inflated?

(None, 2)

Discretized logistic mixture?

(None, 9)

Validation NLL allows to rank different probabilistic models



Flexible Distributions with Transformation Models

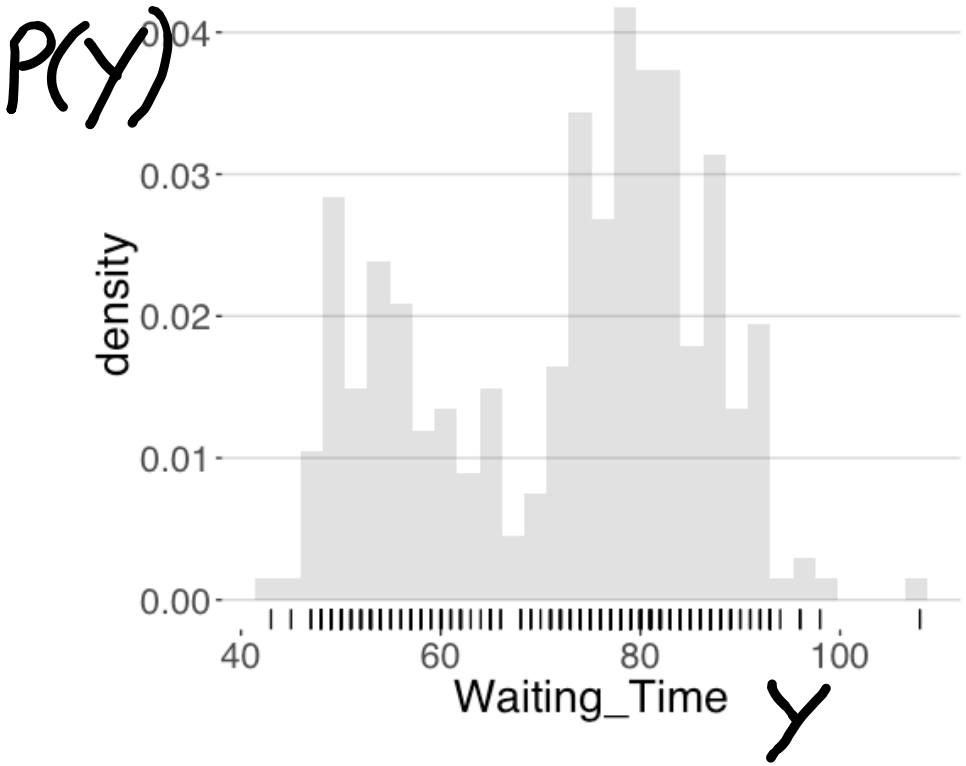
Theory: Name Some Distributions

- Gaussian
- Uniform
- Weibull
- Binomial
- Log-Normal

These are the distributions we have in our Toolbox.

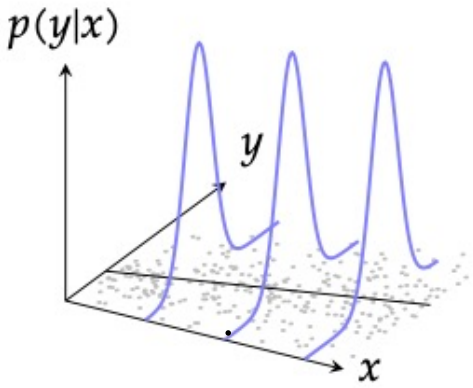
Is the reality like this?

Reality: Data 1-D no covariate (constant x)

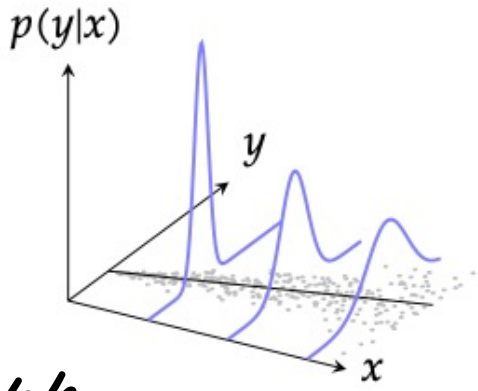


What distribution can you use?

Regression Setting

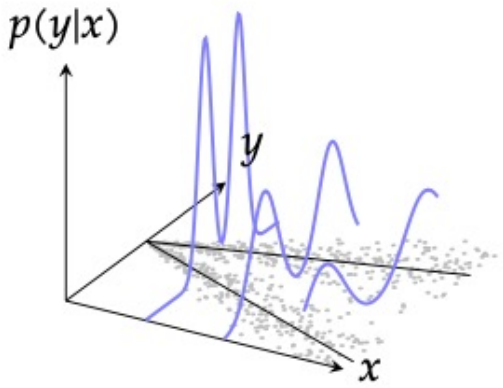


(a) Homoscedastic data, generated by a Gaussian distribution $\mathcal{N}$ with constant variance $\sigma^2 = 2$. The mean is expressed by the linear combination $\mu_x = x + 3$.



Hetero

(b) ~~Homo~~scedastic data, generated by a Gaussian distribution $\mathcal{N}$ with variable variance $\sigma_x^2 = \frac{x}{2}$ and mean $\mu_x = x + 3$.



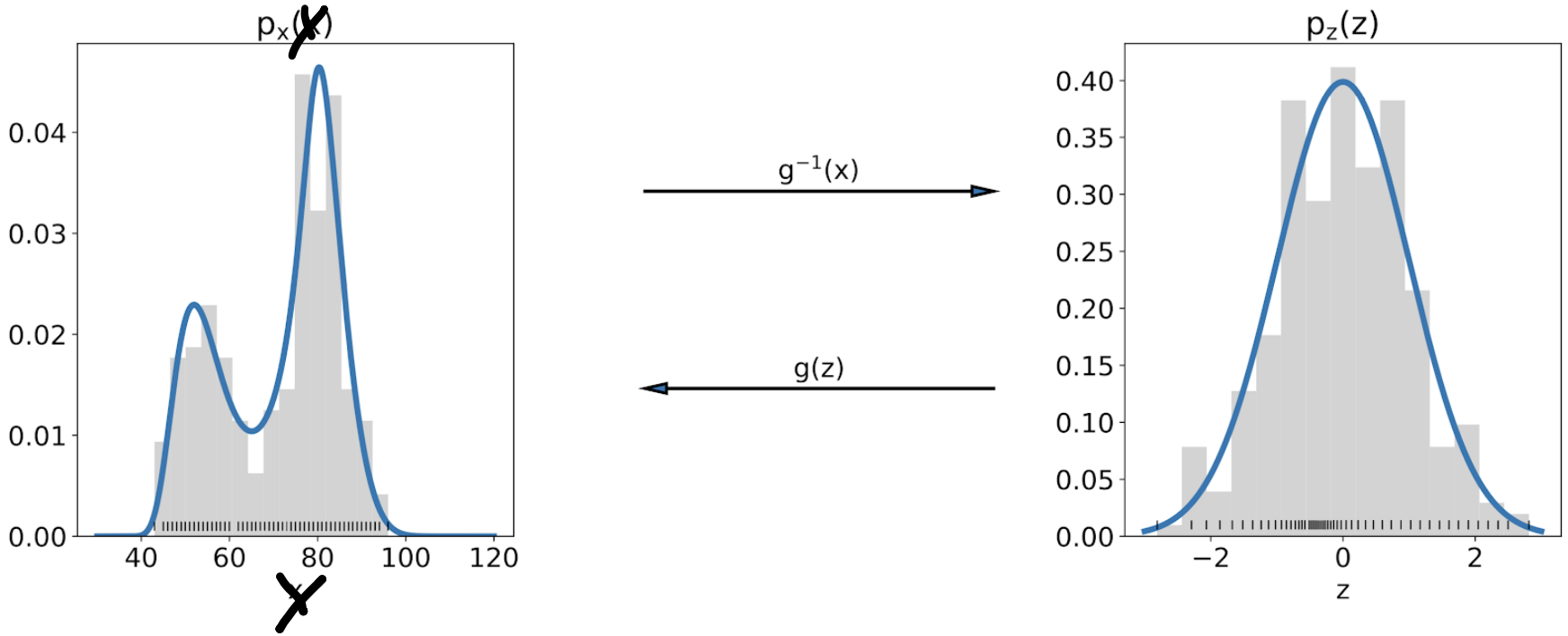
(c) Bimodal data example derived from a mixture of two Gaussian distribution, with the variable variances $\sigma_{1/2,x}^2 = \frac{x}{2}$ and means $\mu_{1,x} = x + 6$ and $\mu_{2,x} = -1.5 \cdot x + 6$.

Figures: MA Thesis Marcel Arpogaus

Idea of Transformation Models in 1D

Data x~strange_function in $\mathbb{R}^1$

Transformed function $z_1 \sim N(0,1)$

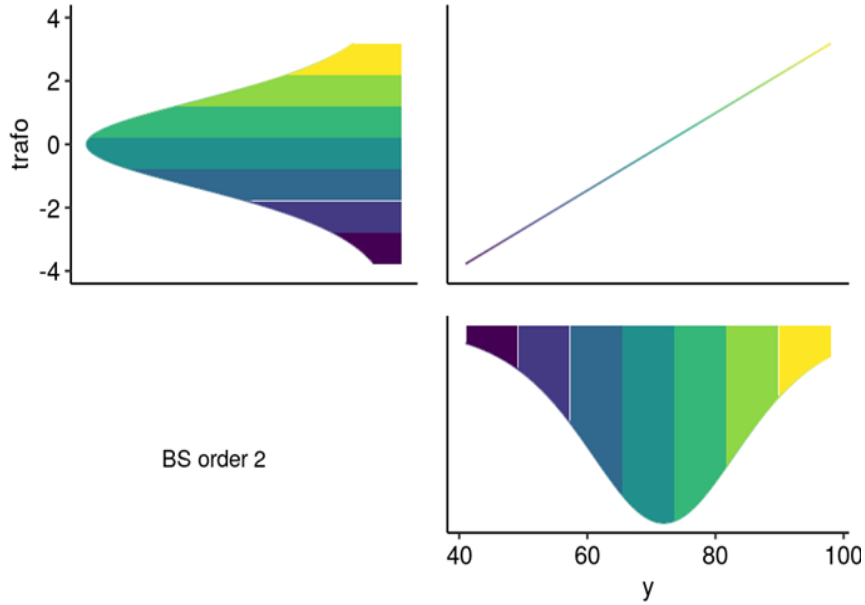


Idea: learn an *invertible* transformation to simple function usually Gaussian $N(0,1)$

- Sampling from $p_x(x)$: sample $z^* \sim p_z(z)$ then transform it via $g_\theta(z)$
- Density of *x**: calculate z*= $g_\theta^{-1}(x^*)$ and evaluate $N(z^*; 0,1)$

There is no fixed naming convention. Also in the direction of g(z) and $g^{-1}(x)$.

Transformation Models Idea



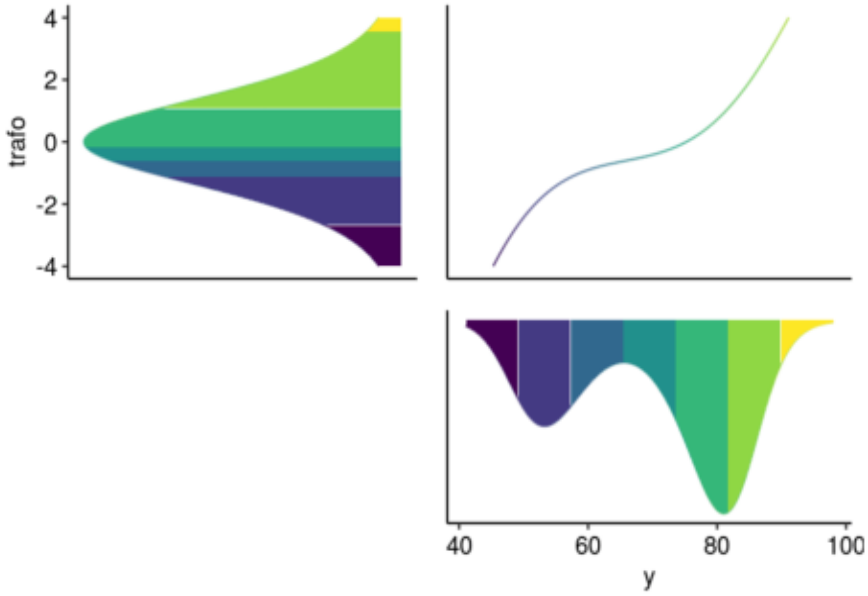
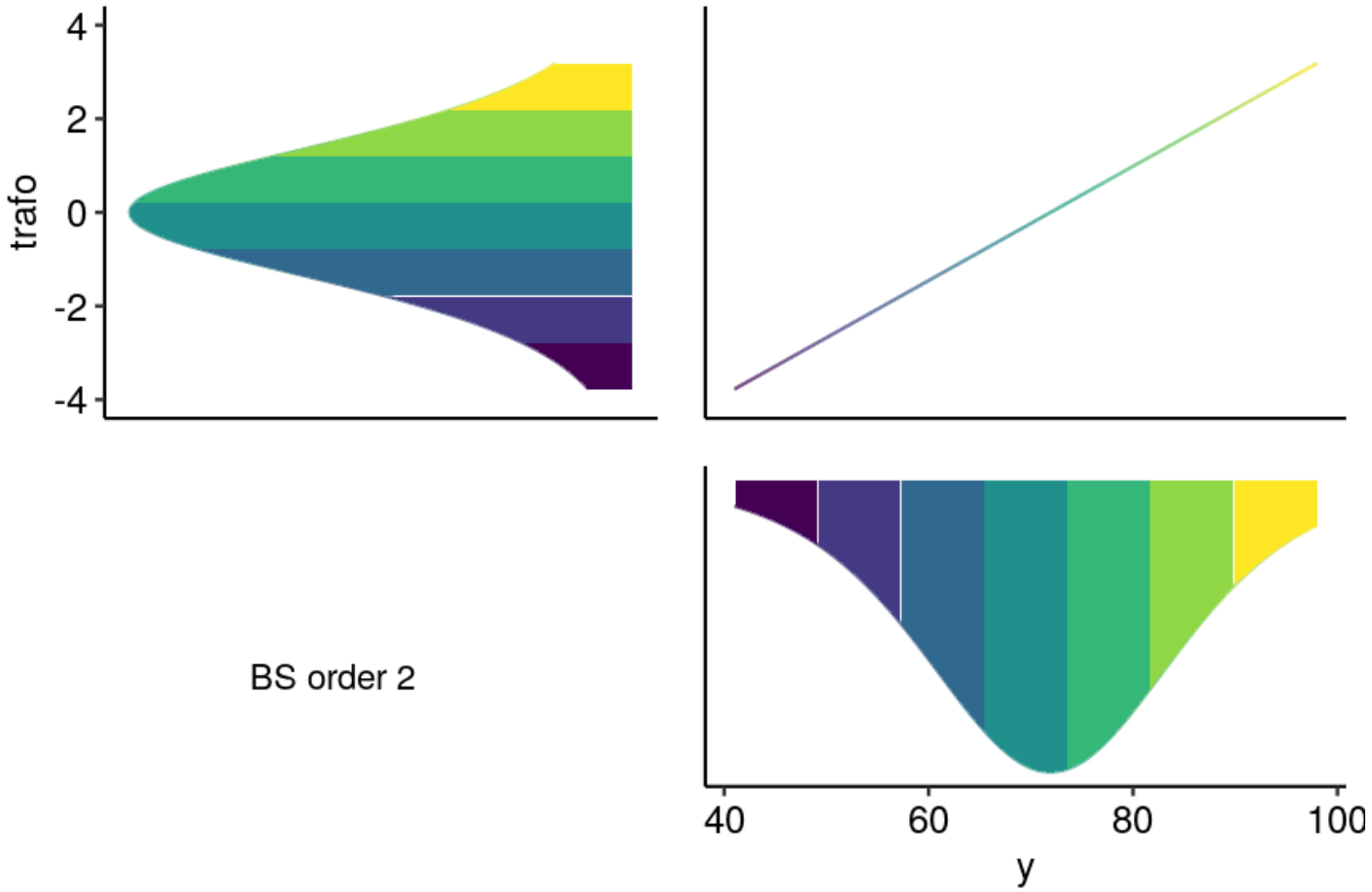


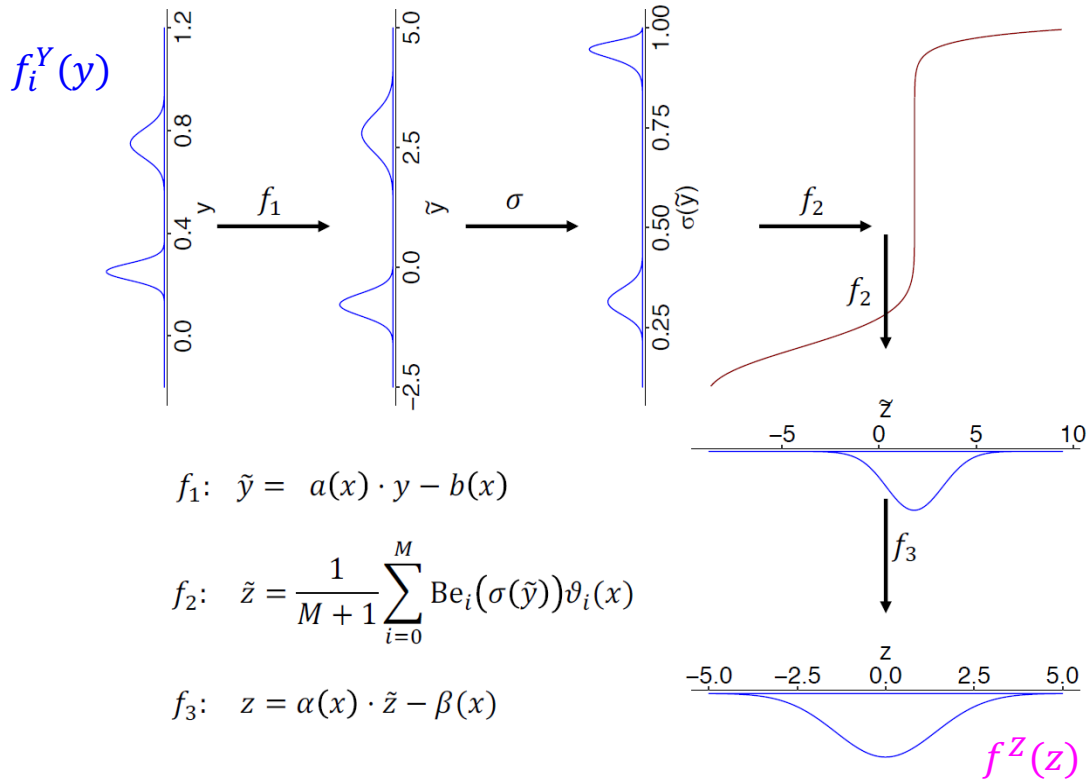
Image credits: Lucas Kook

Transformation Models Simulation



BS order 2

Our deep transformation model



$$f_1: \quad \tilde{y} = a(x) \cdot y - b(x)$$

$$f_2: \quad \tilde{z} = \frac{1}{M+1} \sum_{i=0}^{M} \mathrm{Be}_i\big(\sigma(\tilde{y})\big)\vartheta_i(x)$$

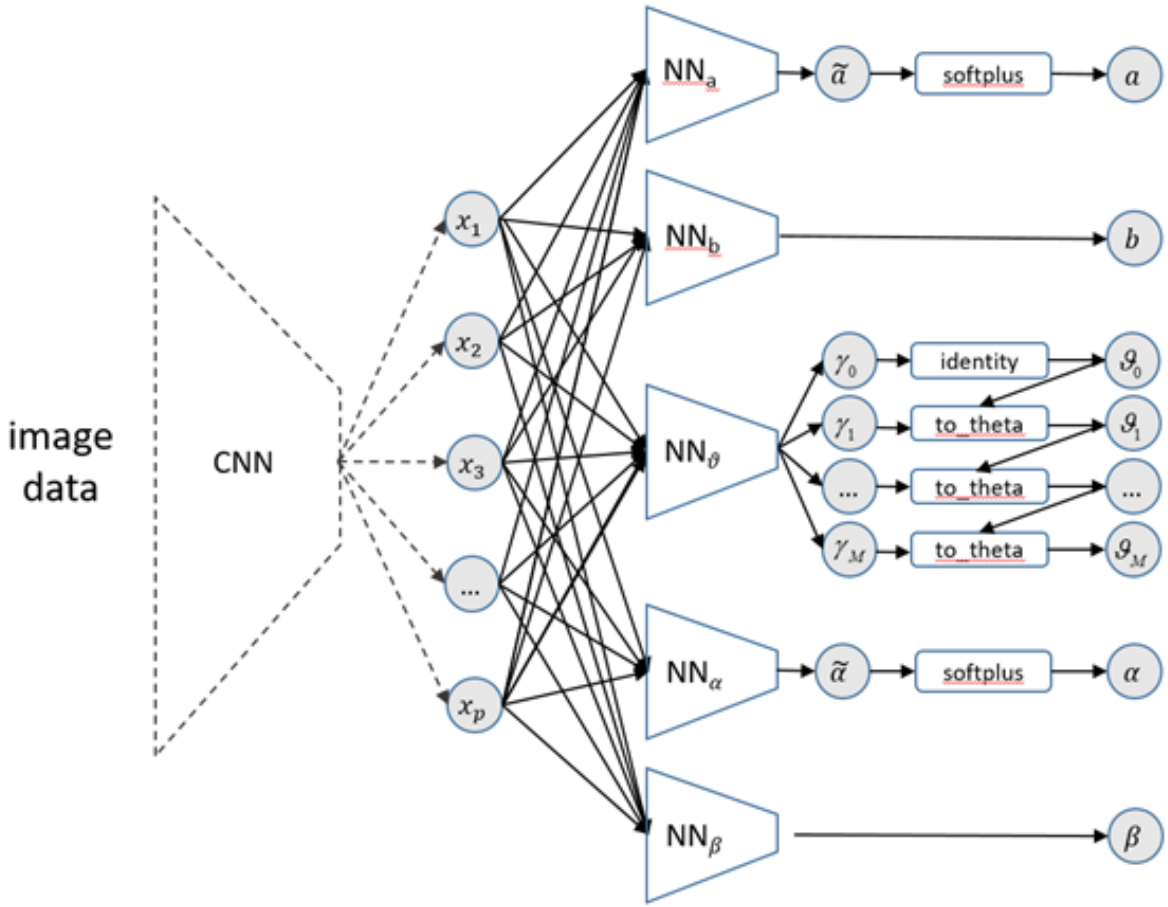
$$f_3: \quad z = \alpha(x) \cdot \tilde{z} - \beta(x)$$

$$h_{\theta(x)} = f_{3,\alpha_x,\beta_x} \circ \; f_{2,\vartheta_{0x},\ldots,\vartheta_{Mx}} \circ \sigma \circ f_{1,a_x,b_x}$$

$$\mathrm{NLL} = \sum_i -\log\left(f^z\,(z_i) \cdot \left| \frac{\partial h_{\widehat{\theta(x_i)}}}{\partial y} \right|\Big|_{y_i} \right)$$

Architecture of our Deep transformation model

$$h_{\theta(x)} = f_{3,\alpha_x,\beta_x} \circ \; f_{2,\vartheta_{0_x},\ldots,\vartheta_{M_x}} \circ \sigma \circ f_{1,a_x,b_x} \qquad \text{NLL} = \sum_i -\log\left(f^z(z_i) \cdot \left|\frac{\partial h_{\widehat{\theta(x_i)}}}{\partial y}\right|\Big|_{y_i}\right)$$



Ensuring a monotone increasing h:

Positive slope:

$a > 0$

Increasing Bernstein coefficients :

$\vartheta_0 = \gamma_0$

$\vartheta_1 = \vartheta_0 + e^{\gamma_1}$

...

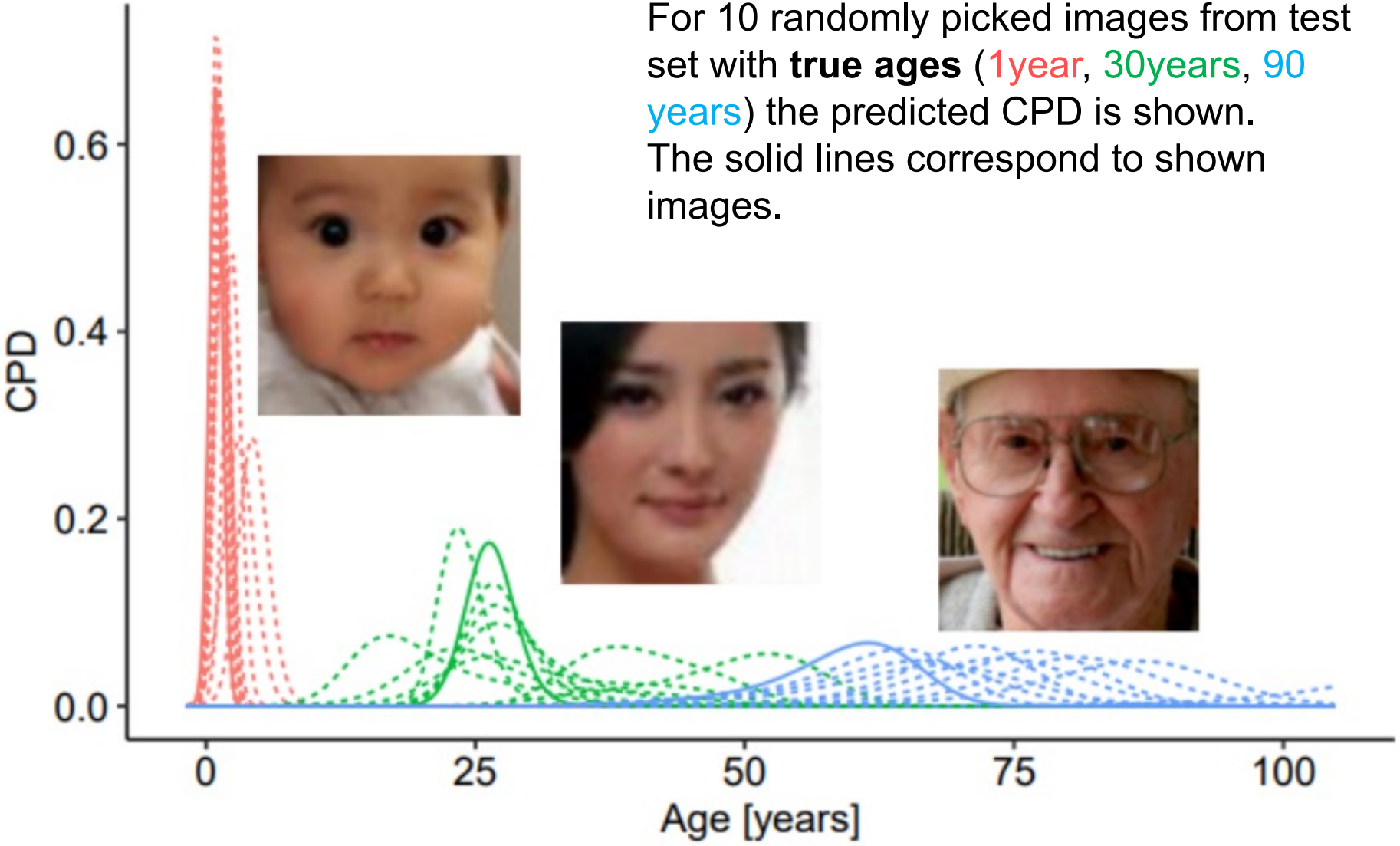
$\vartheta_M = \vartheta_{M-1} + e^{\gamma_M}$

$\rightarrow \vartheta_0 < \vartheta_1 \ldots < \vartheta_M$

Positive slope:

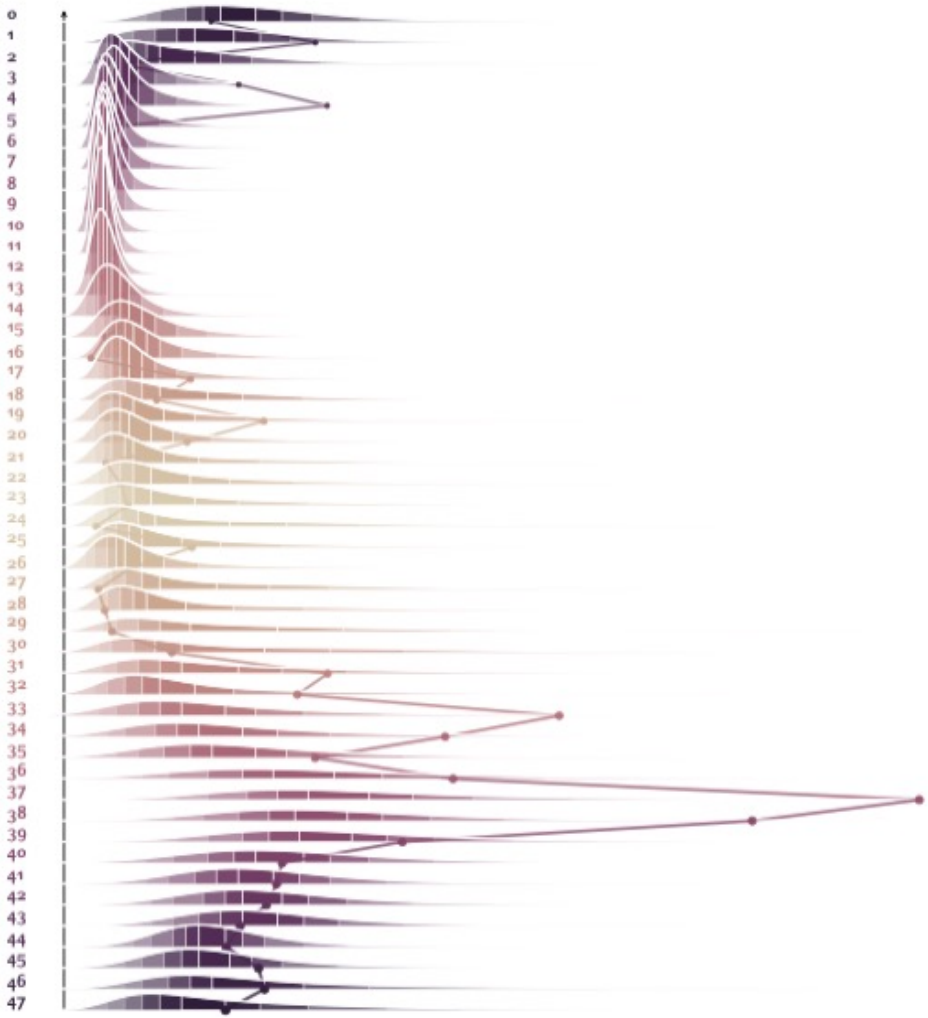
$\alpha > 0$

Application: Predict CPD for age based on an image



For 10 randomly picked images from test set with **true ages** (1year, 30years, 90 years) the predicted CPD is shown. The solid lines correspond to shown images.

Application: Electric Load Consumption Forecast



(a) Ridge plot for the PLF of Saturday from Figure 50.

Architecture	Probabilistic model	Score
Feedforward	Normal distribution	-9.57
	Gaussian mixture	-90.01
	Normalizing flow	**-98.75**
WaveNet	Normal distribution	-7.65
	Gaussian mixture	-94.14
	Normalizing flow	**-96.09**

Deep Transformation model (further resources)

- R implementation:
  - https://github.com/tensorchiefs/dl_mlt
- Python implementation:
  - https://github.com/MArpogaus/TensorFlow-Probability-Bernstein-Polynomial-Bijector
- Paper
  - https://arxiv.org/abs/2004.00464

Take home messages CPD

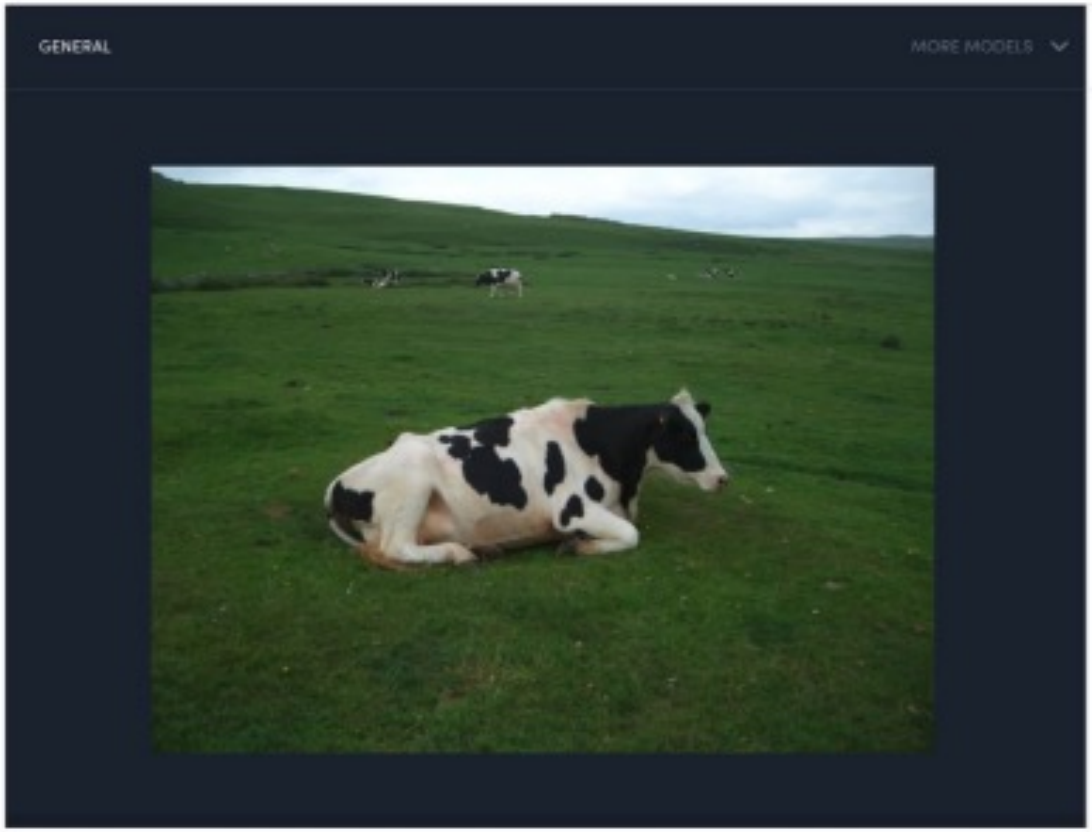
- A probabilistic model predicts for each input a whole outcome CPD
- Probabilistic models are the models of choice when dealing with uncertainty
- Probabilistic models fit naturally in DL framework
- Good Probabilistic modeling improves SOTA in deep learning models
- Use the NLL for **training**, **evaluating** and **comparing** probabilistic models

Bayesian Deep Learning

Outline

- Issues with current DL approach
  - No extrapolating / no epistemic uncertainty
- Bayesian Statistics
  - A simple example (Bayes the Hacker's way)
  - Introduction to Bayesian Statistics
- Bayesian Neural Networks
  - Variational Approximation
  - MC-Dropout

Typisches Beispiel einer Kuh

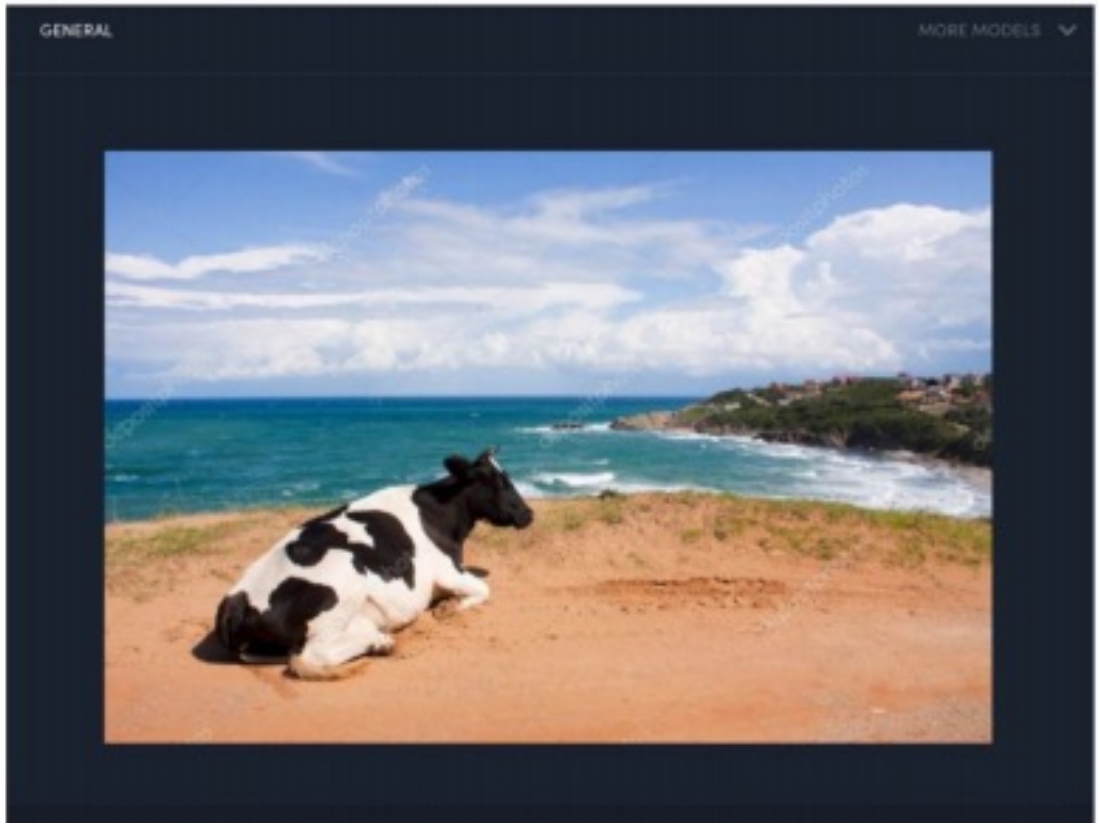


General	VIEW DOCS
cow	0.992
cattle	0.983
mammal	0.979
grass	0.978
livestock	0.966
farm	0.964
landscape	0.963
pasture	0.954
grassland	0.949
agriculture	0.948
no person	0.945

NeurIPS 2018 Workshop on Causal Learning

Untypisches Beispiel einer Kuh

Dieses Bild ist zu weit weg von den Trainingsdaten. Kuh nicht unabhängig von Hintergrund



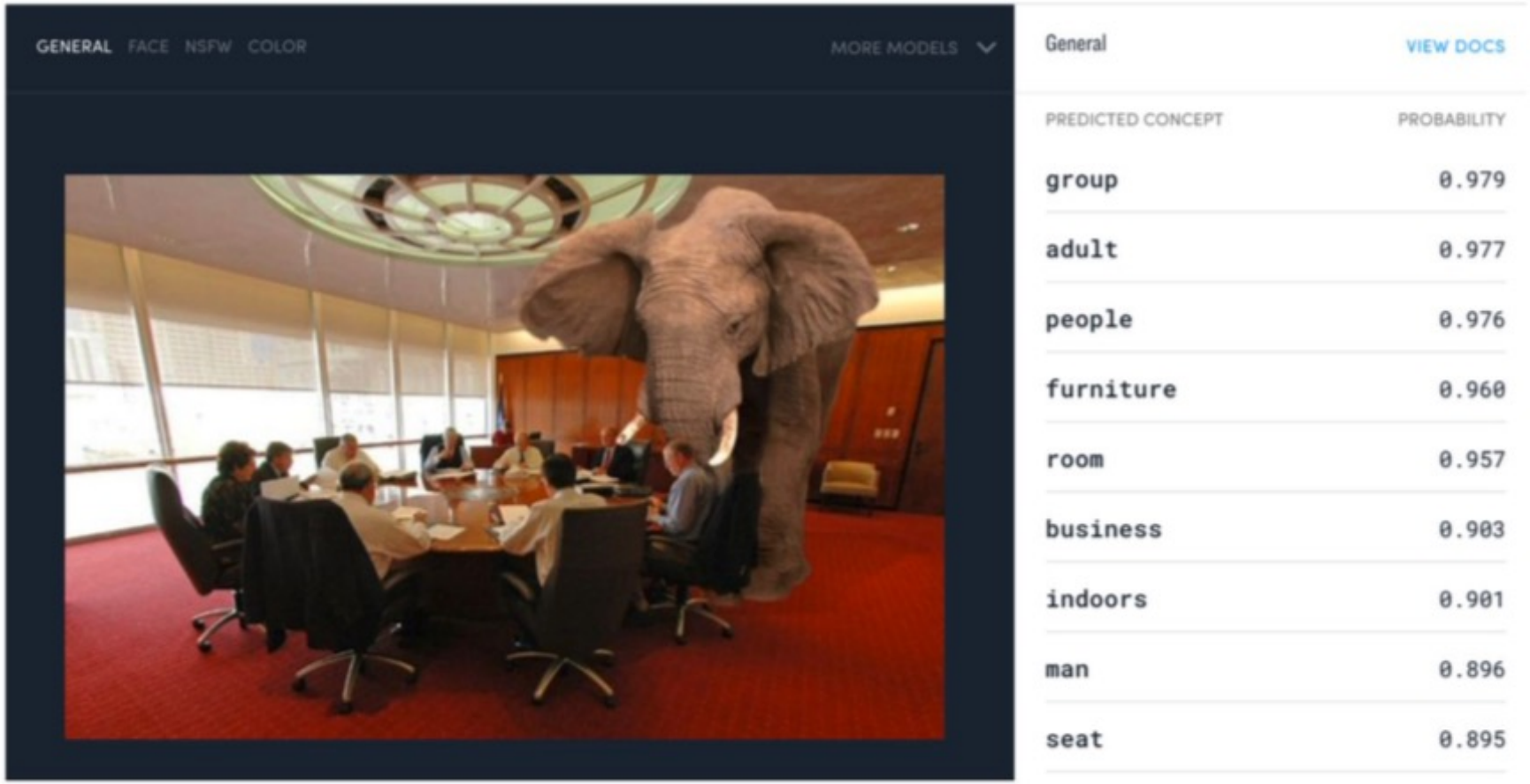
Neuronales Netz erkennt keine Kuh!

General	VIEW DOCS
no person	0.991
beach	0.990
water	0.985
sand	0.981
sea	0.980
travel	0.978
seashore	0.972
summer	0.954
sky	0.946
outdoors	0.944
ocean	0.936

NeurIPS 2018 Workshop on Causal Learning

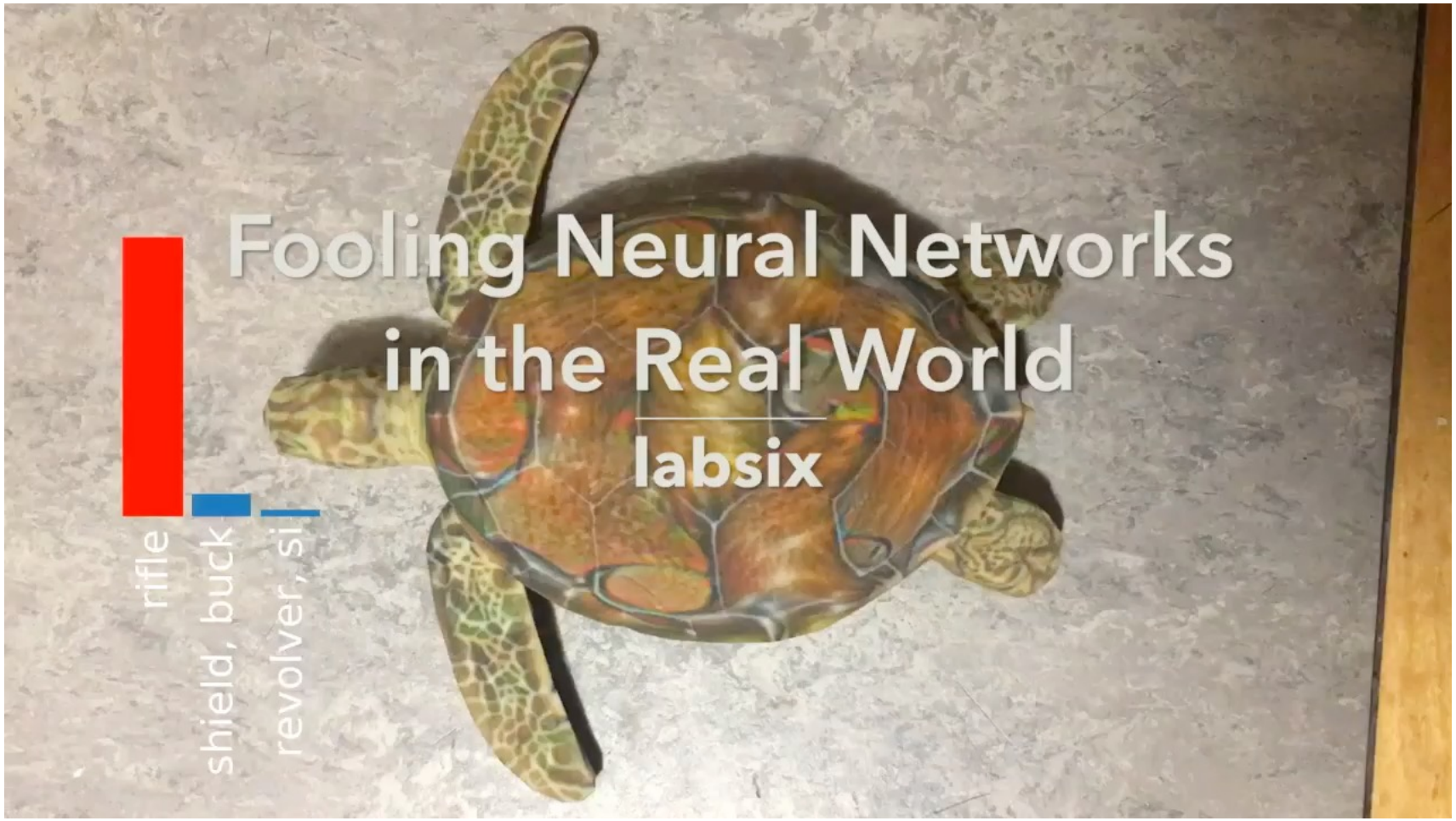
The elephant in the room

Neuronales Netz erkennt keinen Elefanten!



NeurIPS 2018 Workshop on Causal Learning

Turtle as Rifle

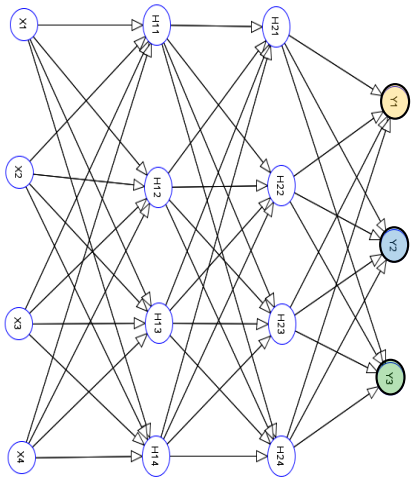


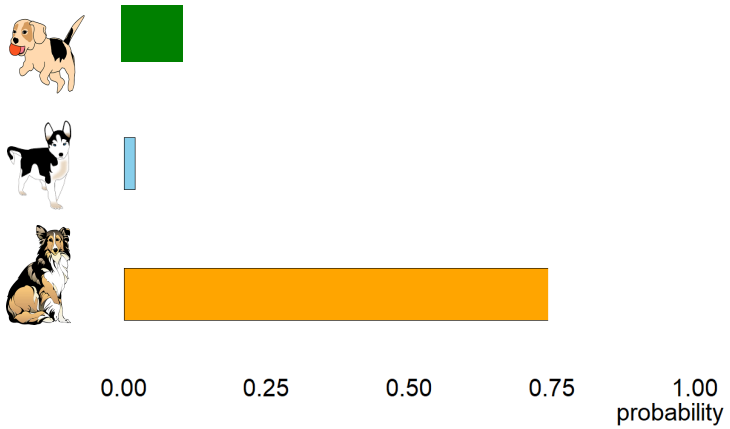
Ausgedruckes Exemplar

Source: https://www.labsix.org/physical-objects-that-fool-neural-nets/

What do we get from a DL classification model?

Suppose you train a classifier to discriminate between 3 dog breeds





The prediction is “collie” because it gets the highest probability: p_{pred}=0.75

What is 0.75 telling us.

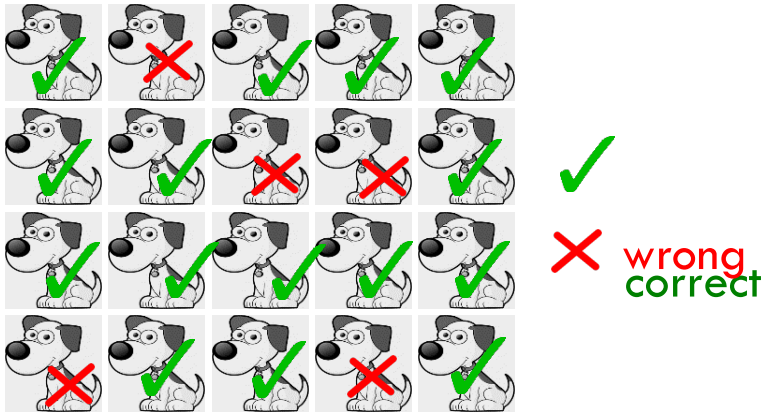
What is the probability telling?

p_{pred}=0.75

Among many predications that had p_{pred}=0.75, we expect that on average 75% of these predictions are correct and only 25% predictions are wrong

→ Then the classifier produces *calibrated probabilities*

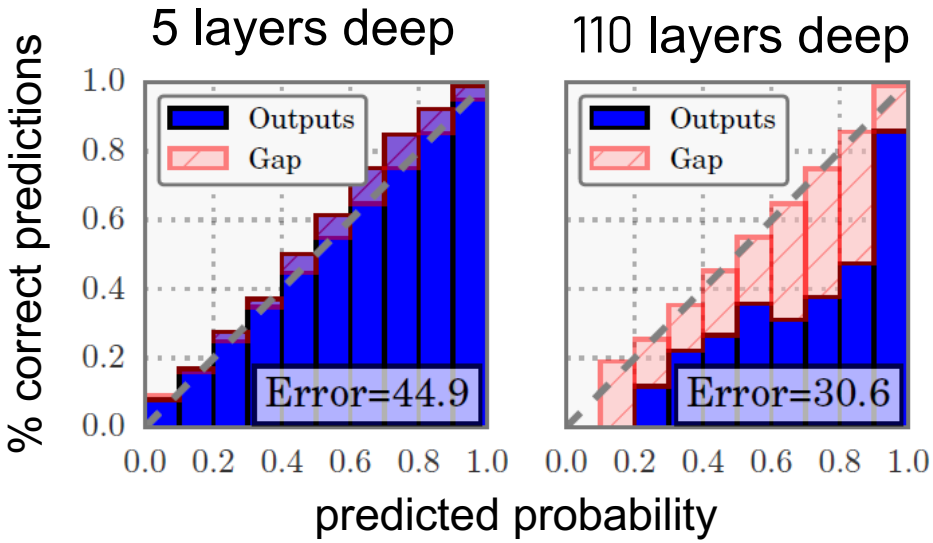
Sample of images where the predicted class got p=0.75:



Do CNNs produce calibrated probabilities?

Guo et al. (2017)
On Calibration of Modern NN

The deeper CNNs get
- the fewer miss-classifications
- the less well calibrated they get

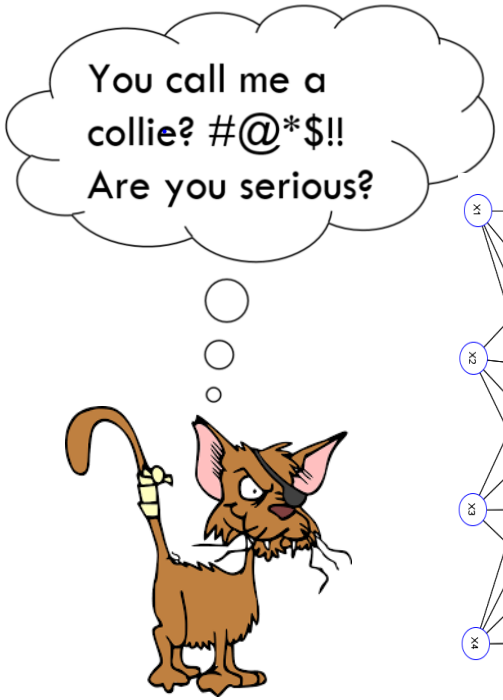


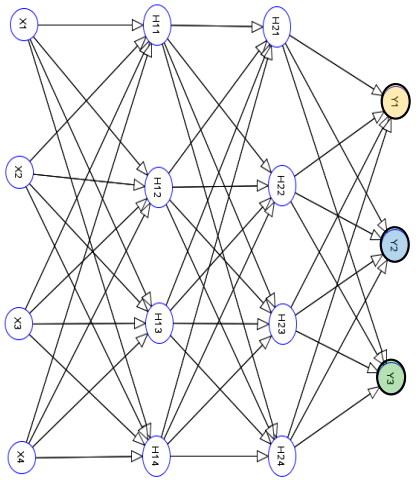
Good news:
deep NN can be “recalibrated” and then we get calibrated probabilities.

62

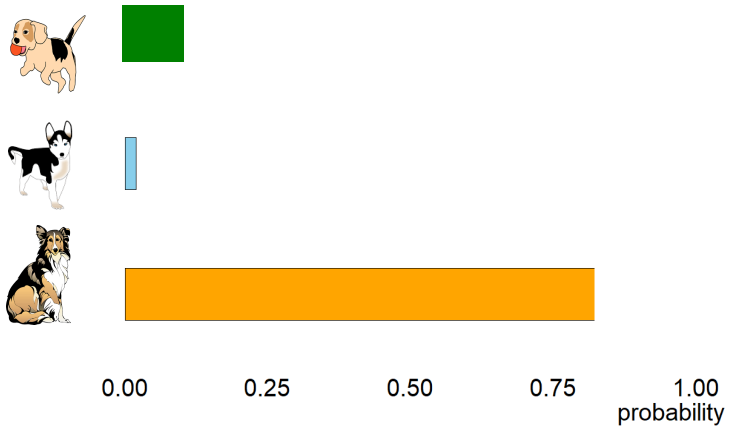
CNNs yield high accuracy
and calibrated probabilities, but…

What happens if a novel class is presented to the CNN?





Plain wrong !



The reported probability is only valid if, we have the P(Train)=P(Test).
That's not the case. "The big lie deep learning is based on".
This is more evident, if we have a look at regression problems.

Elephant in the room

- Aufgabe: https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/18_elephant_in_the_room.ipynb

Extrapolation

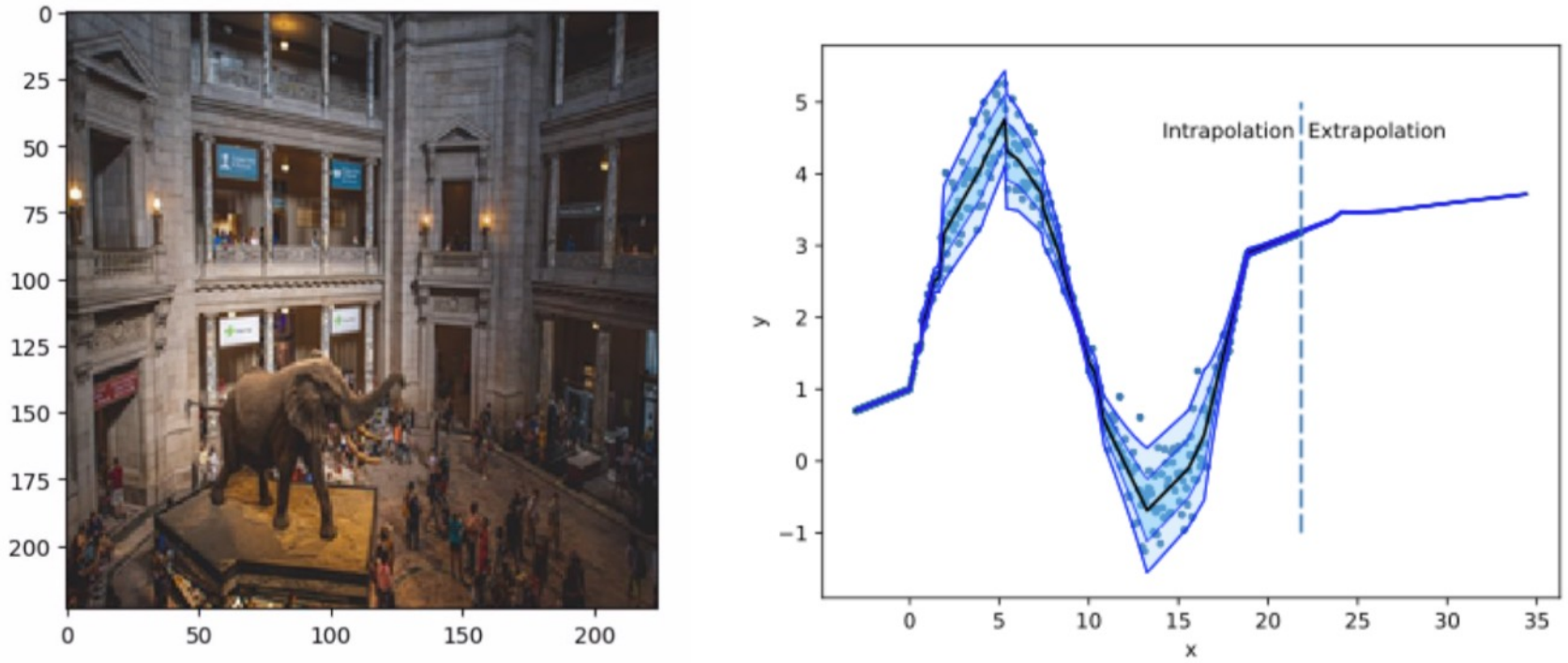
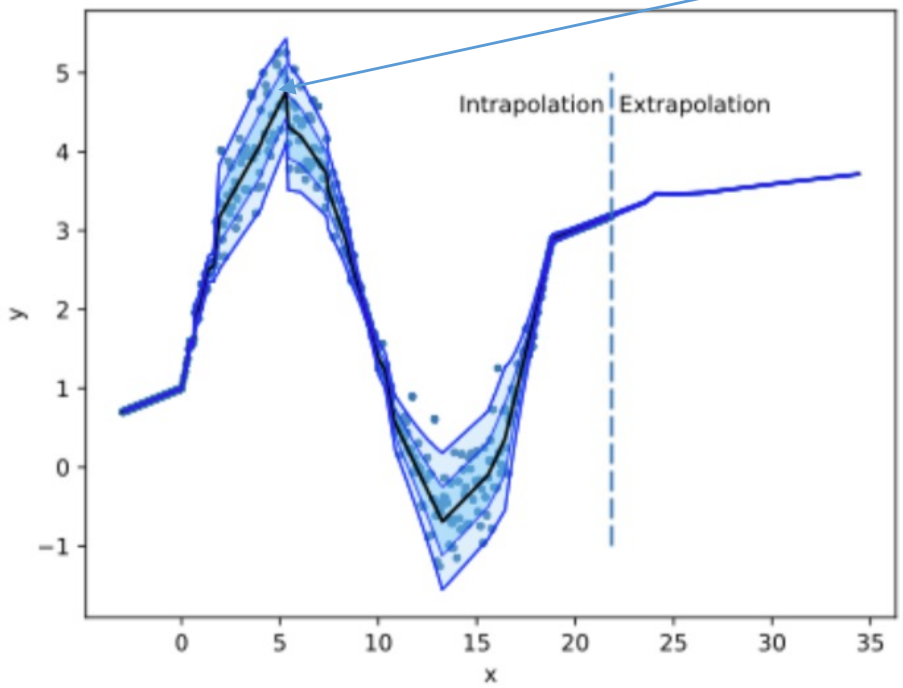


Figure 7.2 Bad case of DL. The high performant VGG16-CNN trained on ImageNet data fails to see the elephant in the room. The five highest-ranked class predictions of the objects in the image are horse_cart, shopping_cart, palace, streetcar, gondola; the elephant is not found! This image is an extrapolation of the training set. In the regression problem on the right side of the dashed vertical line, there's zero uncertainty in the regions where there's no data (extrapolation).

Aleatroic vs. Epistemic Uncertainty

Much spread in data, aleatroic (from “Alea Acta est”))



- *Aleatoric* uncertainty is due to the uncertainty in the data.
- The uncertainty when leaving the ‘known ground’ is called *epistemic* uncertainty.

We can model this uncertainty when we take the uncertainty with which we know the weights (called parameter uncertainty) into account. This can be done with Bayesian reasoning.