Machine Intelligence:: Deep Learning
Week 6

Beate Sick, Oliver Dürr, Jonas Brändli

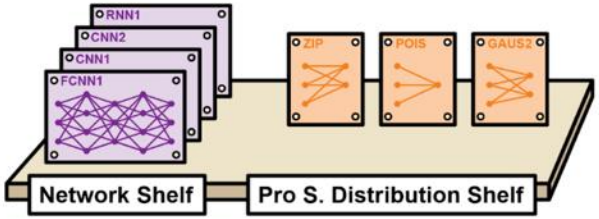
Probablistic models with flexible CPDs

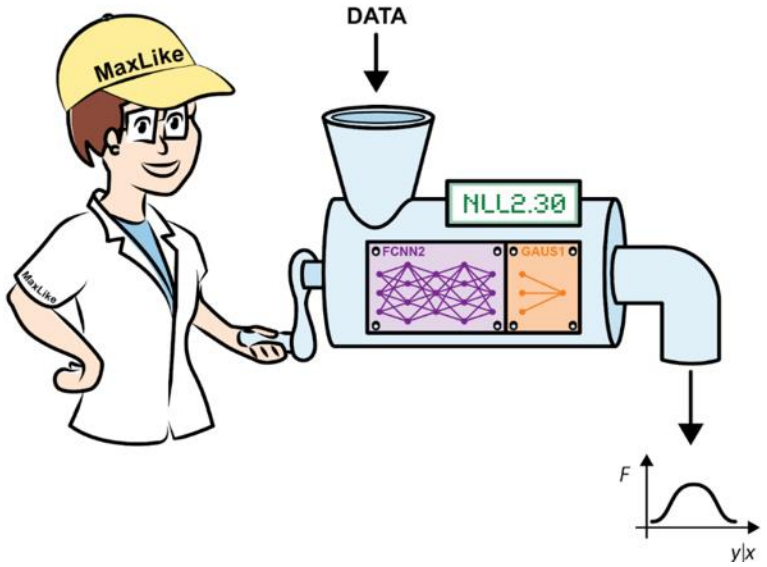
Outline of the DL Module (tentative)

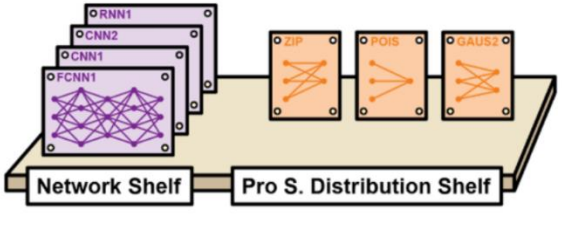
- Day 1: Jumpstart to DL
  - What is DL
  - Basic Building Blocks
  - Keras
- Day 2: CNN I
  - ImageData
- Day 3: CNN II and RNN
  - Tips and Tricks
  - Modern Architectures
  - 1-D Sequential Data
- Day 4: Looking at details
  - Linear Regression
  - Backpropagation
    - Resnet
  - Likelihood principle
- 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 conditional distributions
  - Grad CAM
  - Deep Ensembles
- Day 7: Uncertainty in DL
  - Bayesian Modeling
- Day 8: Uncertainty cont'd
  - Bayesian Neural Networks
  - Projects

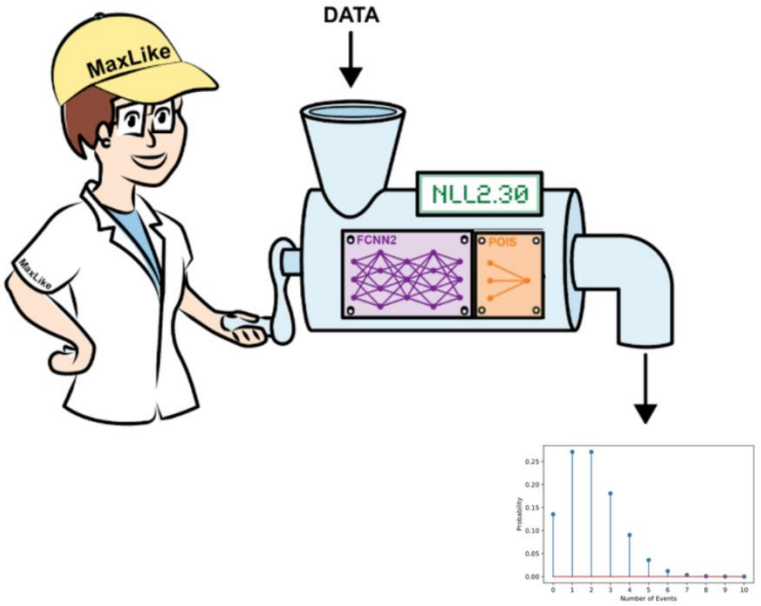
Projects please register (see website)
https://docs.google.com/spreadsheets/d/18VFrPbKq3YSOg8Ebc1q1wGgkfgaWl7IkcCClGEDGj6Q/edit#gid=0

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

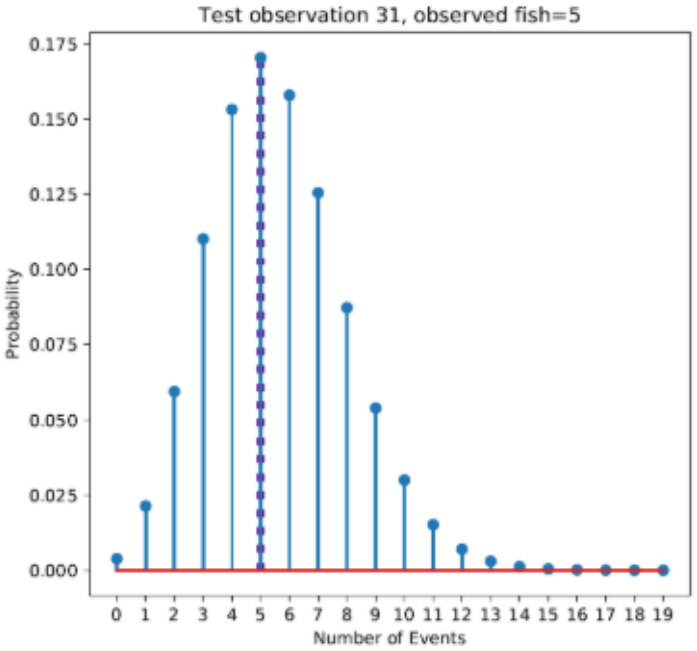
X4=camper: indicates if camper is brought

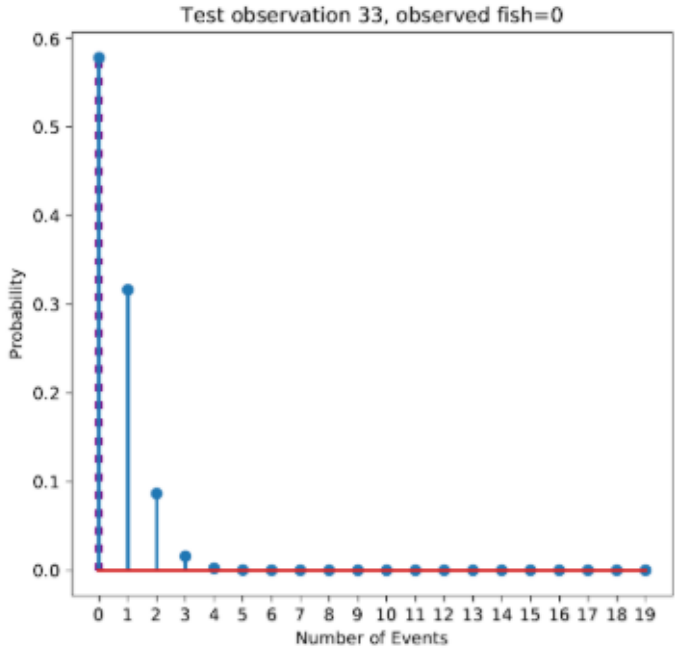
Data: https://stats.idre.ucla.edu/r/dae/zip

Model 2: Poisson regression, predicted CPDs for test observations

Predict CPD for outcome in test data:

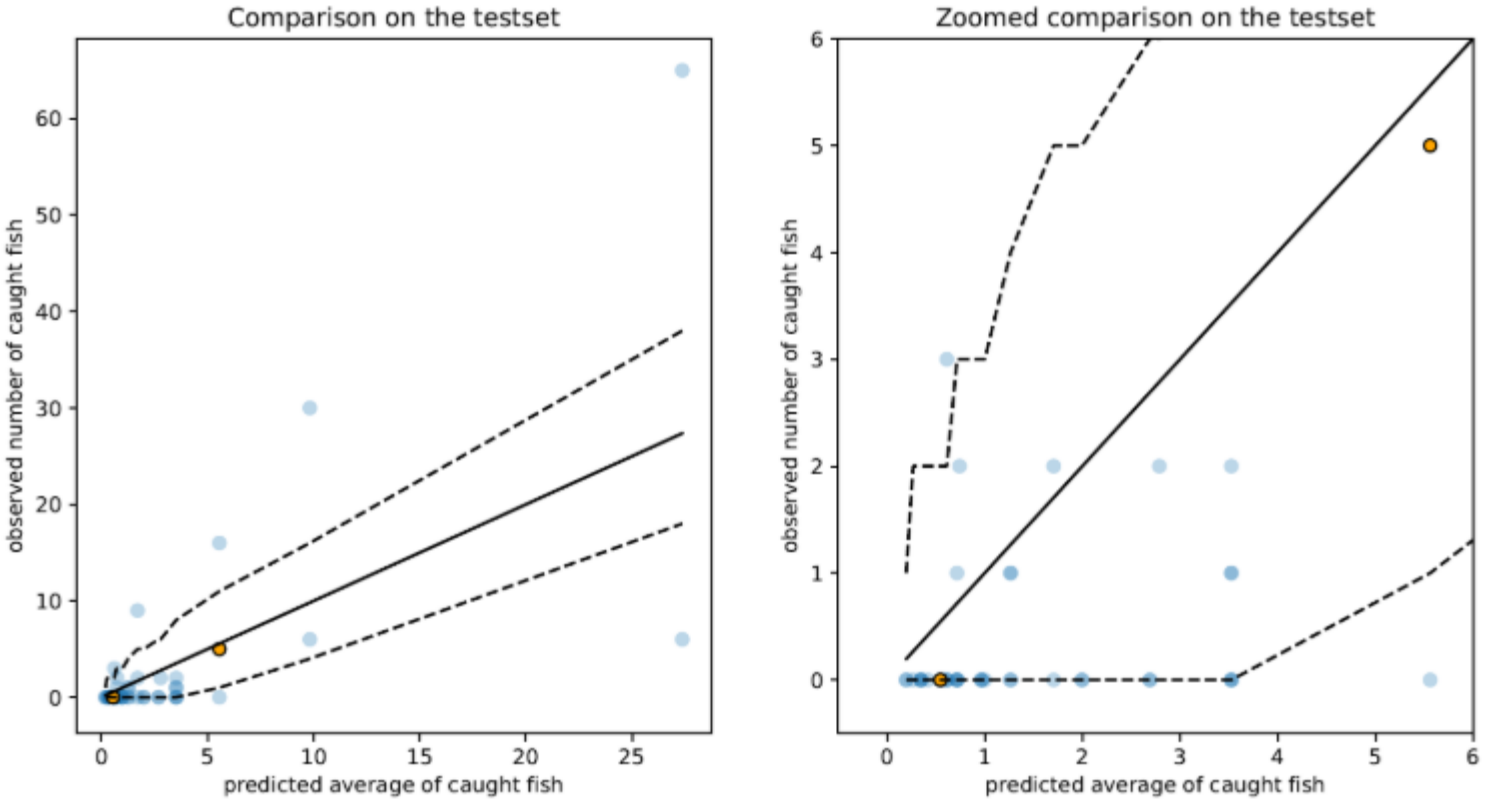
Group 31 used livebait, had a camper and were 4 persons with one child. Y=5 fish.
Group 33 used livebait, didn't have a camper and were 4 persons with two childern. Y=0 fish.





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

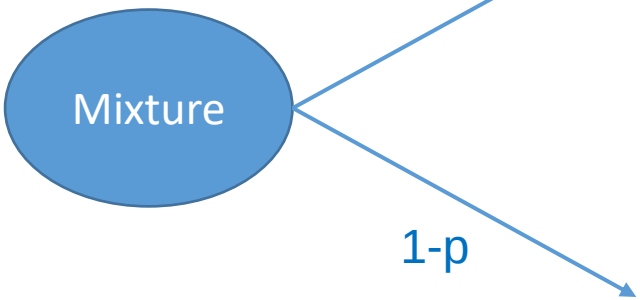
https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/14_poisreg_with_tfp.ipynb

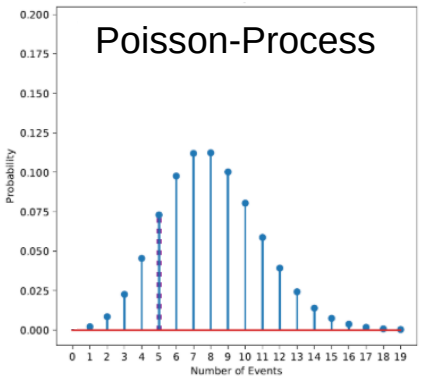
Modeling count data:

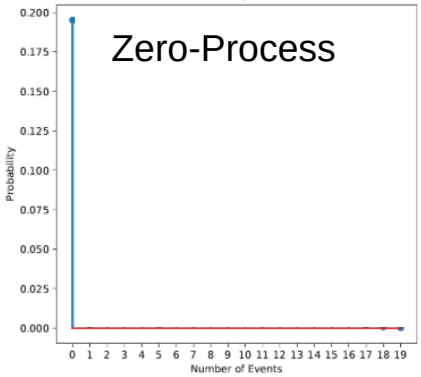
M3: ZIP regression

Zero-Inflated Poisson (ZIP) as Mixture Process

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



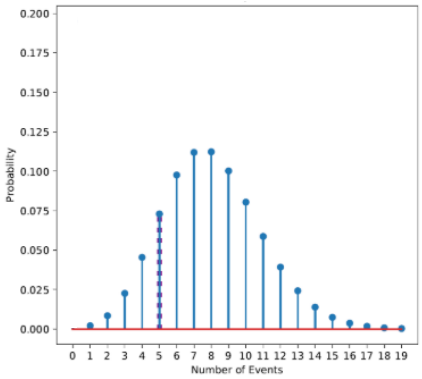




Zero-Inflated Poisson (ZIP) can be seen as Mixture Distribution

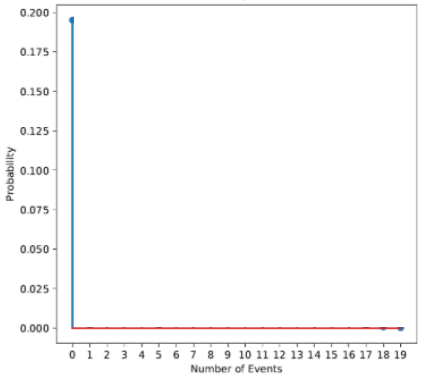
Poisson-Process

$p \cdot$



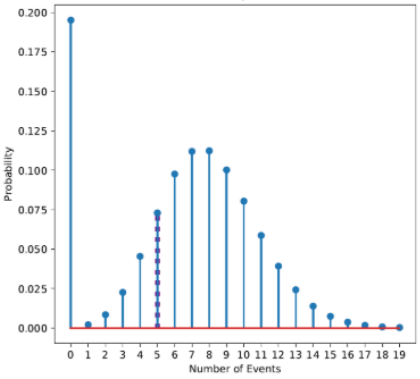
$+(1\text{-}p) \cdot$

Zero-Process

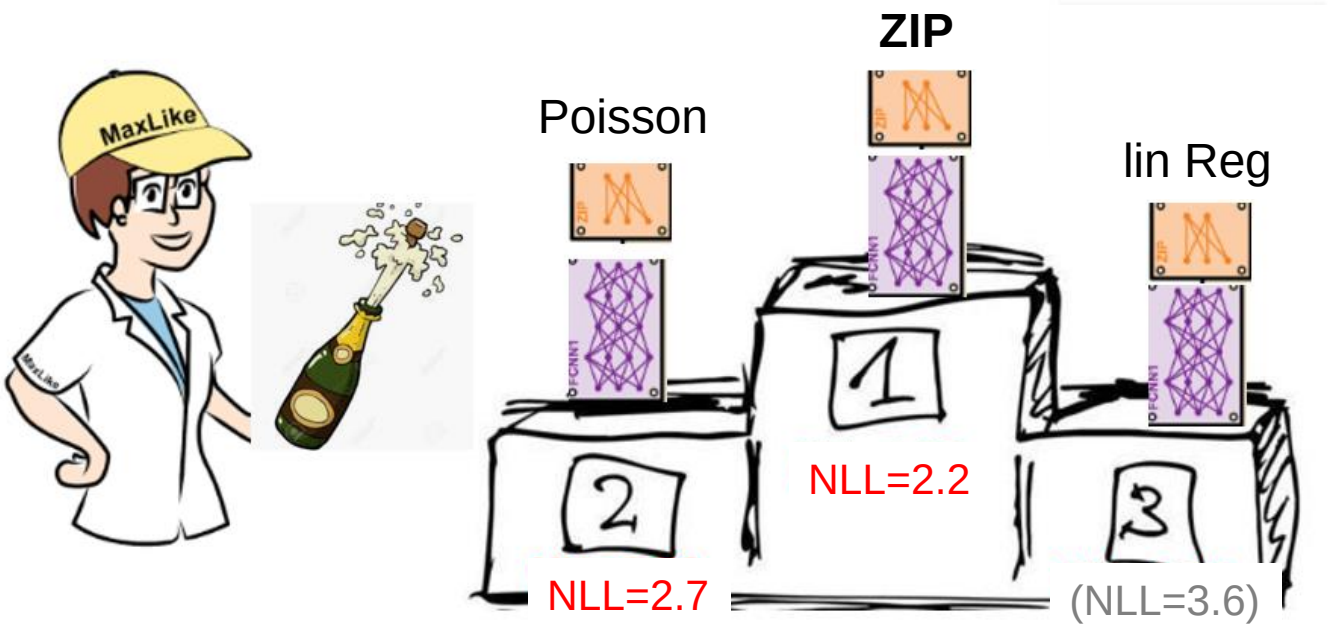


$=$

Zero-inflated Poisson



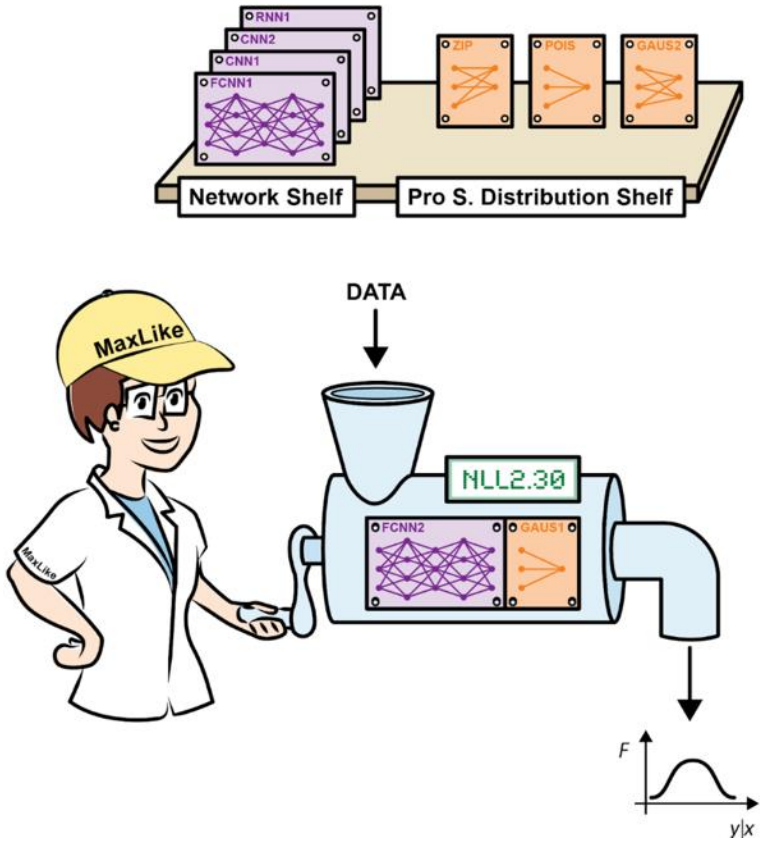
Validation NLL allows to rank different probabilistic models



We cant compare NLLs from discrete and continuous models

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

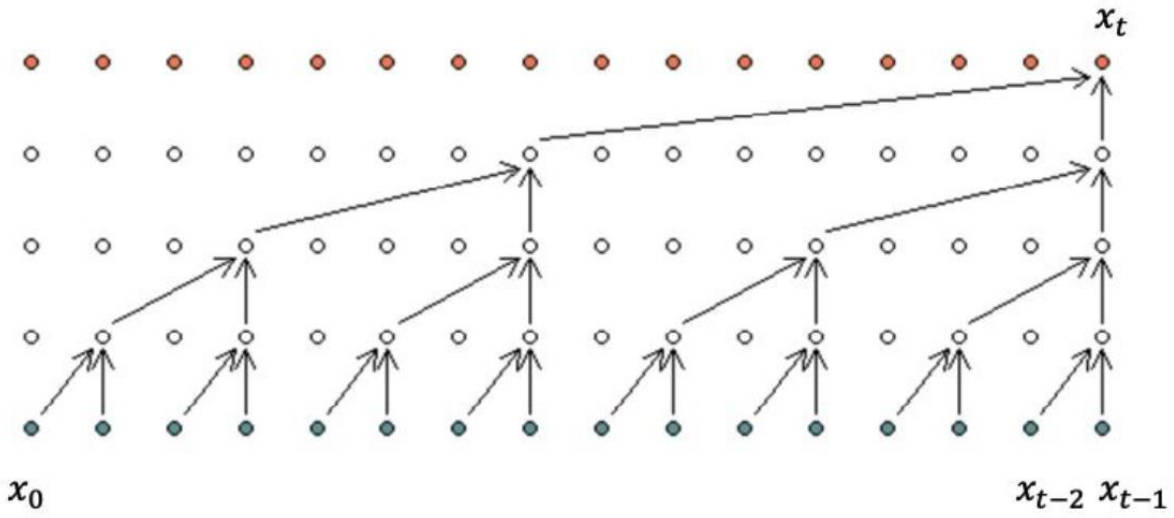
- Train with NLL
- Evaluate performance on validation with NLL

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

- 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 65536 (2^16)
  - 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

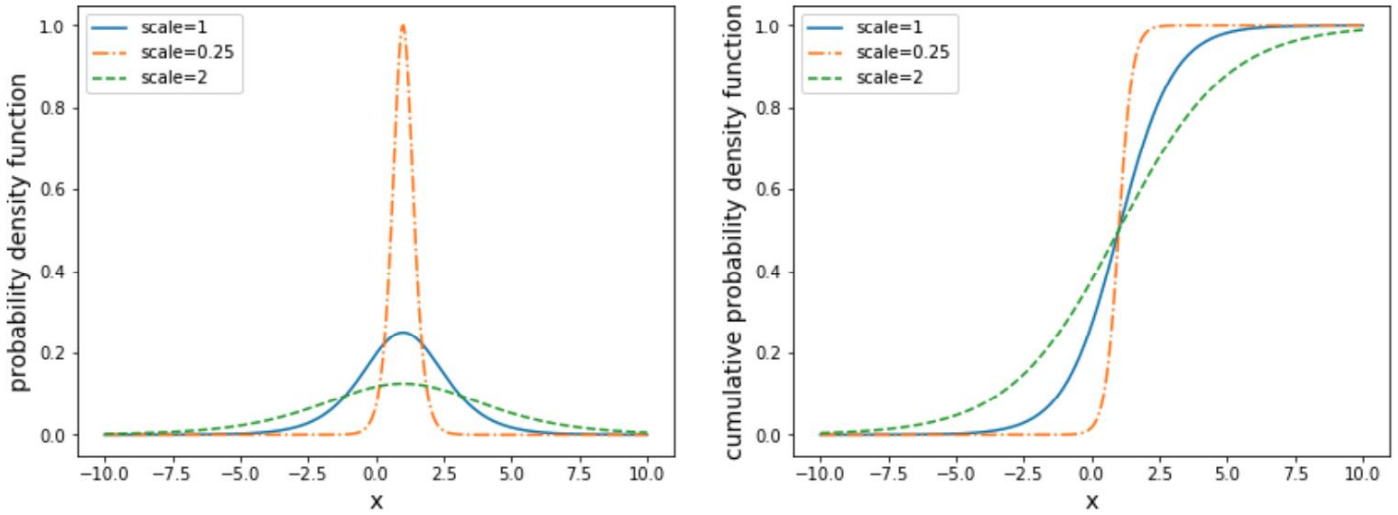
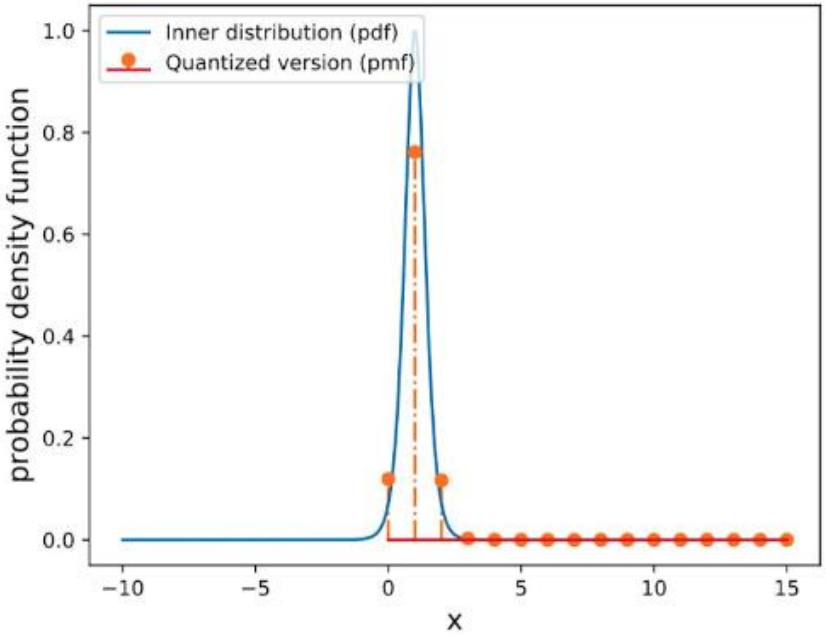


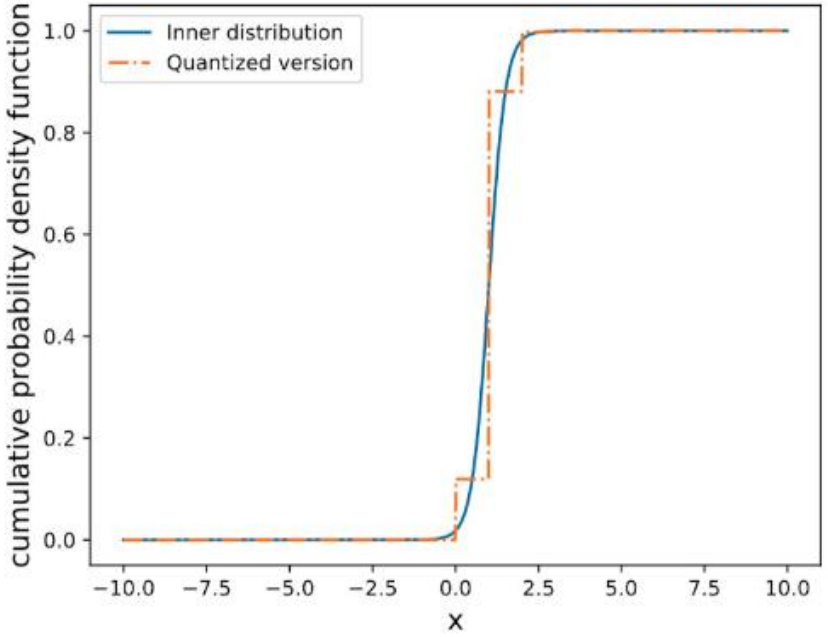
Figure 6.3 Three logistic functions created using tfd.Logistic(loc=1, scale=scale) with values of 0.5, 1.0, and 2.0 for the scale parameter. On the left is the probability density function (PDF) and on the right, the cumulative probability density function (CDF).

- 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

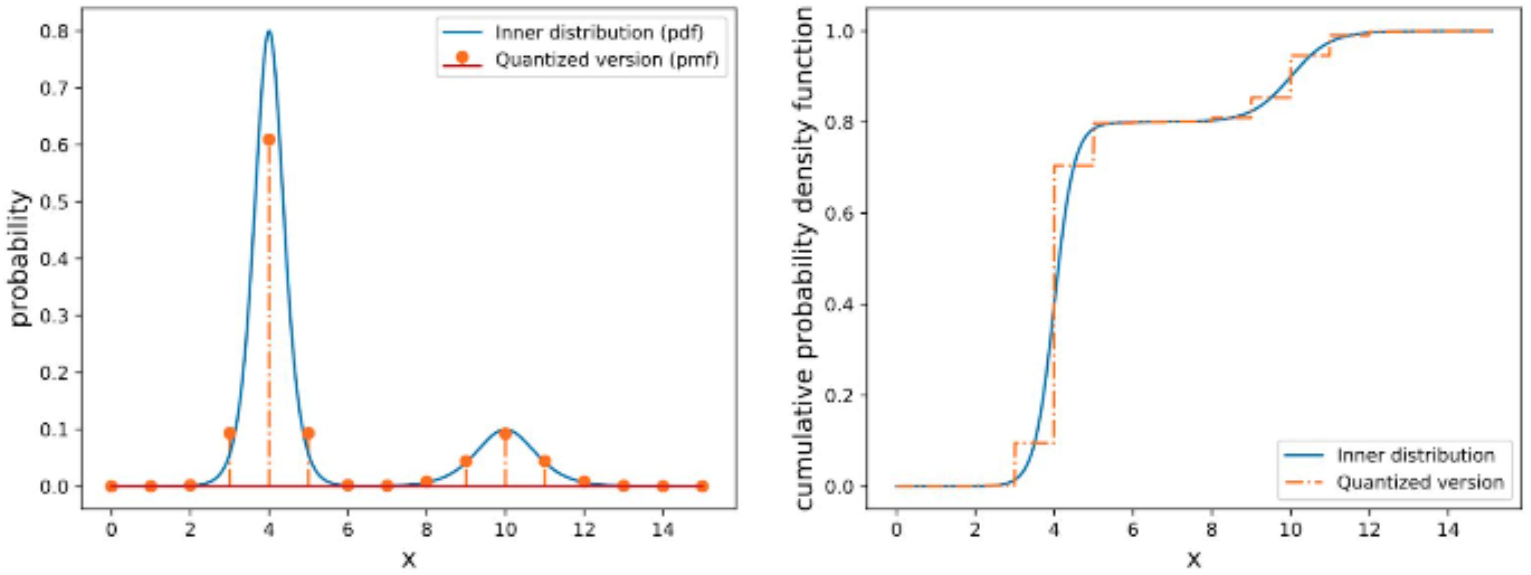


Figure 6.5 The resulting discrete distribution when mixing two logistic distributions (see listing 6.2 for the code that produces these plots)

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

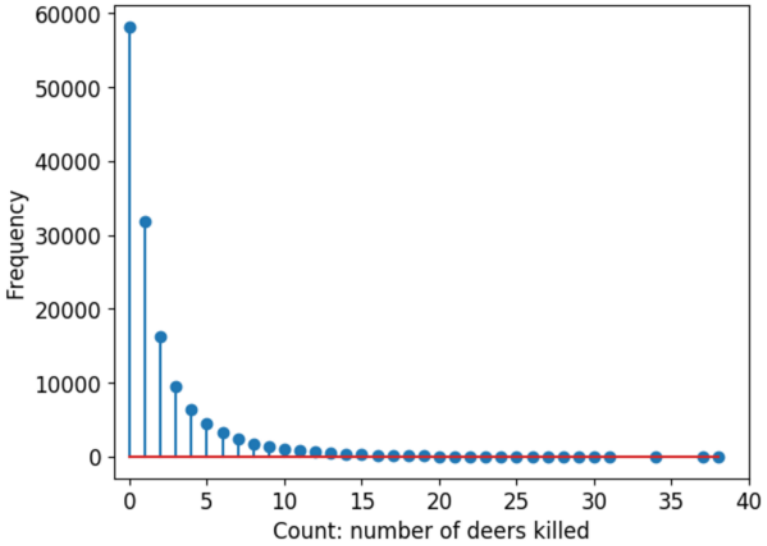
Goal:
Predict CPD for y=#deers-killed-in-30min, given x (time and derived variables).
Possible CPD models:

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

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

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

Marginal distribution of deer killed in 30min



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
_________________________________________________________________
```

This depends CPD

Poisson

(None, 1)

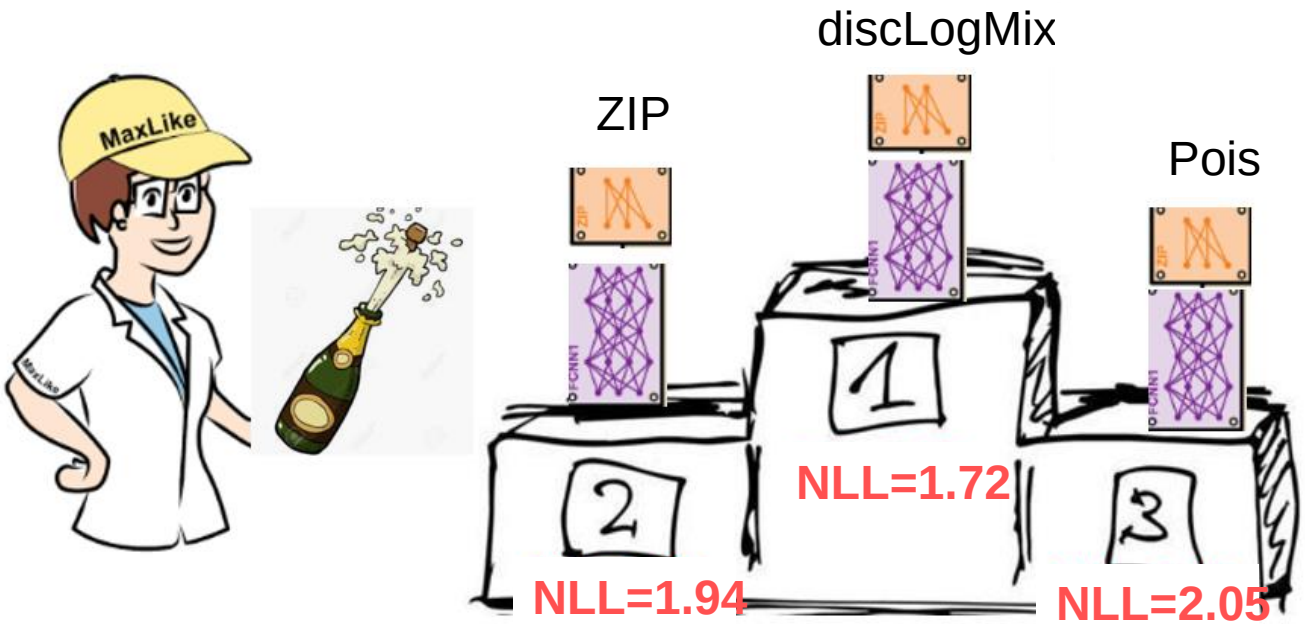
Zero Inflated?

(None, 2)

Discretized logistic mixture?

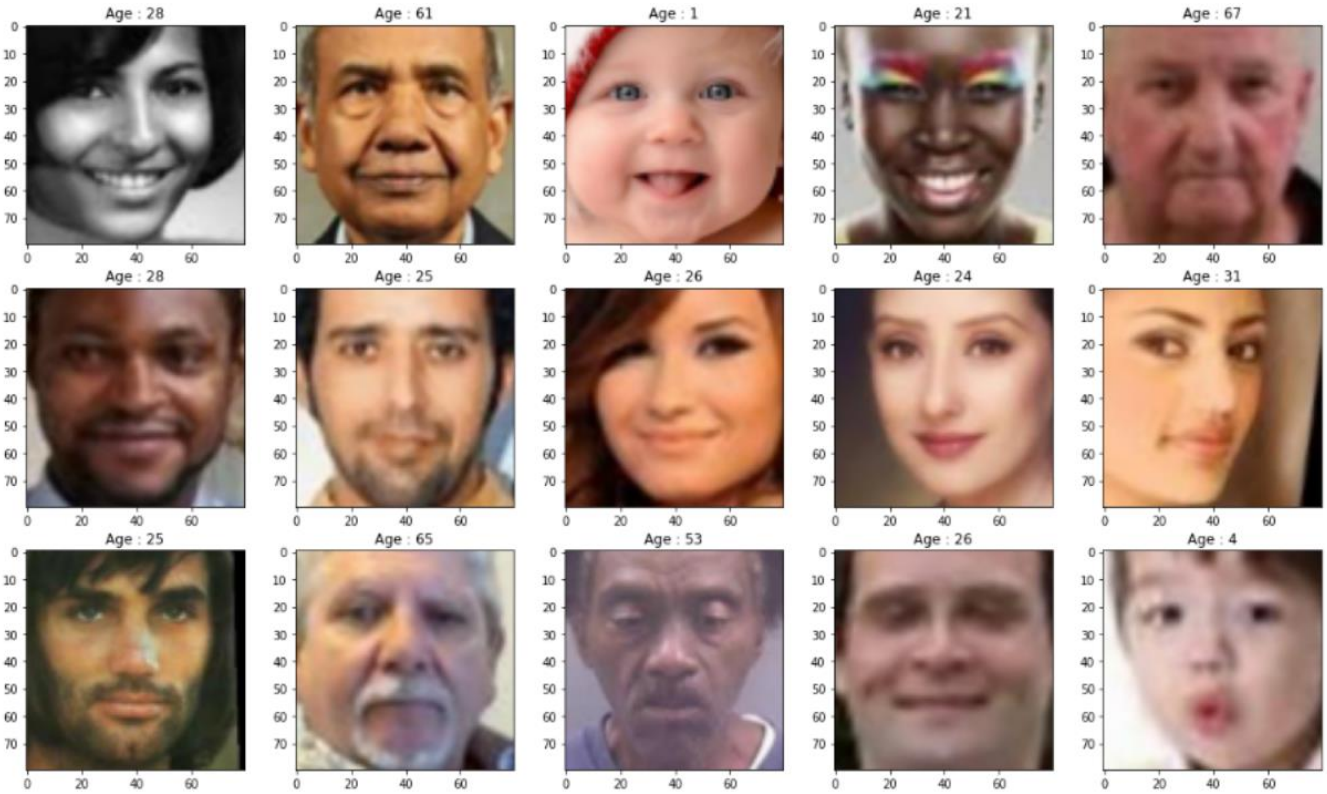
(None, 9)

Validation NLL allows to rank different probabilistic models



Probabilistic models with complex input data

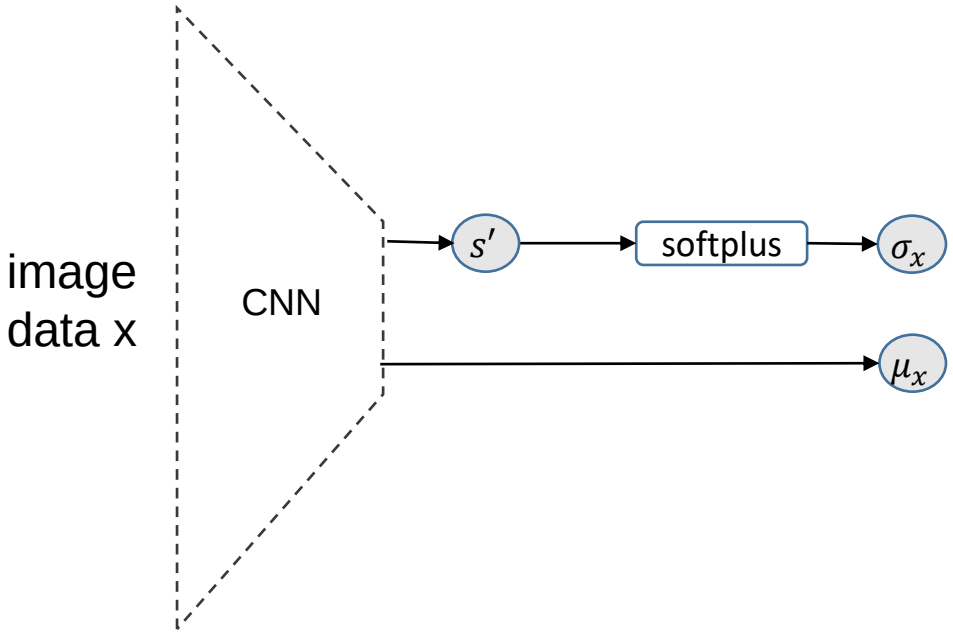
The UTK face data – face image data with known age



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):

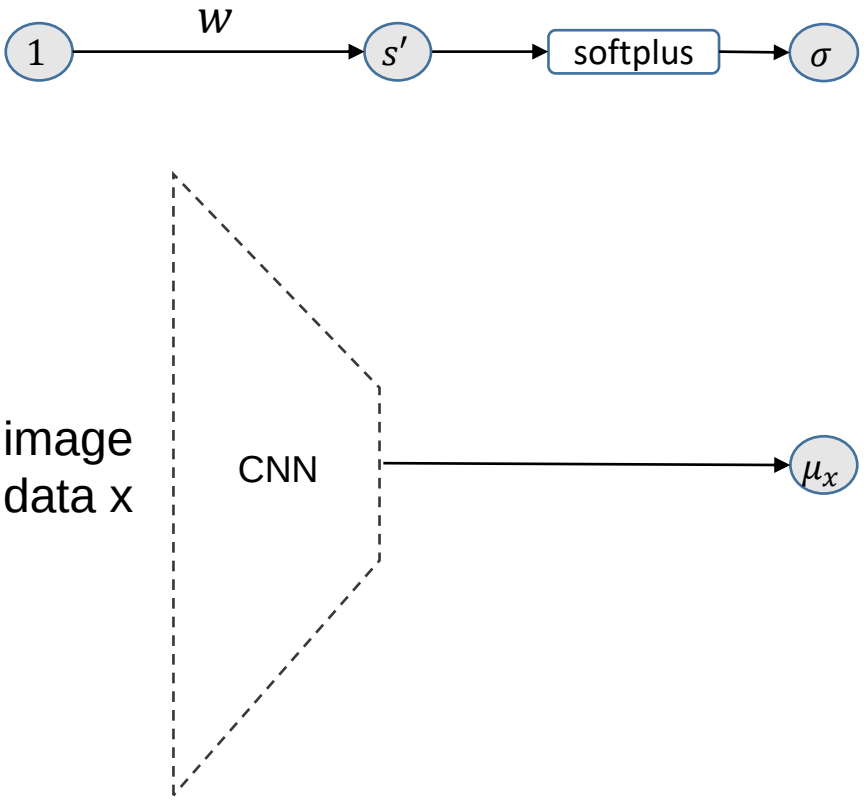
$$\hat{\mathbf{w}}_{\mathrm{ML}} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma^{2}}}\right) + \frac{\left(y_i - \mu_{x_i}\right)^2}{2\sigma^2}$$

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 σ =constant

Modeling Gaussian CPD with flexible mean & constant variance



Minimize the negative log-likelihood (NLL):

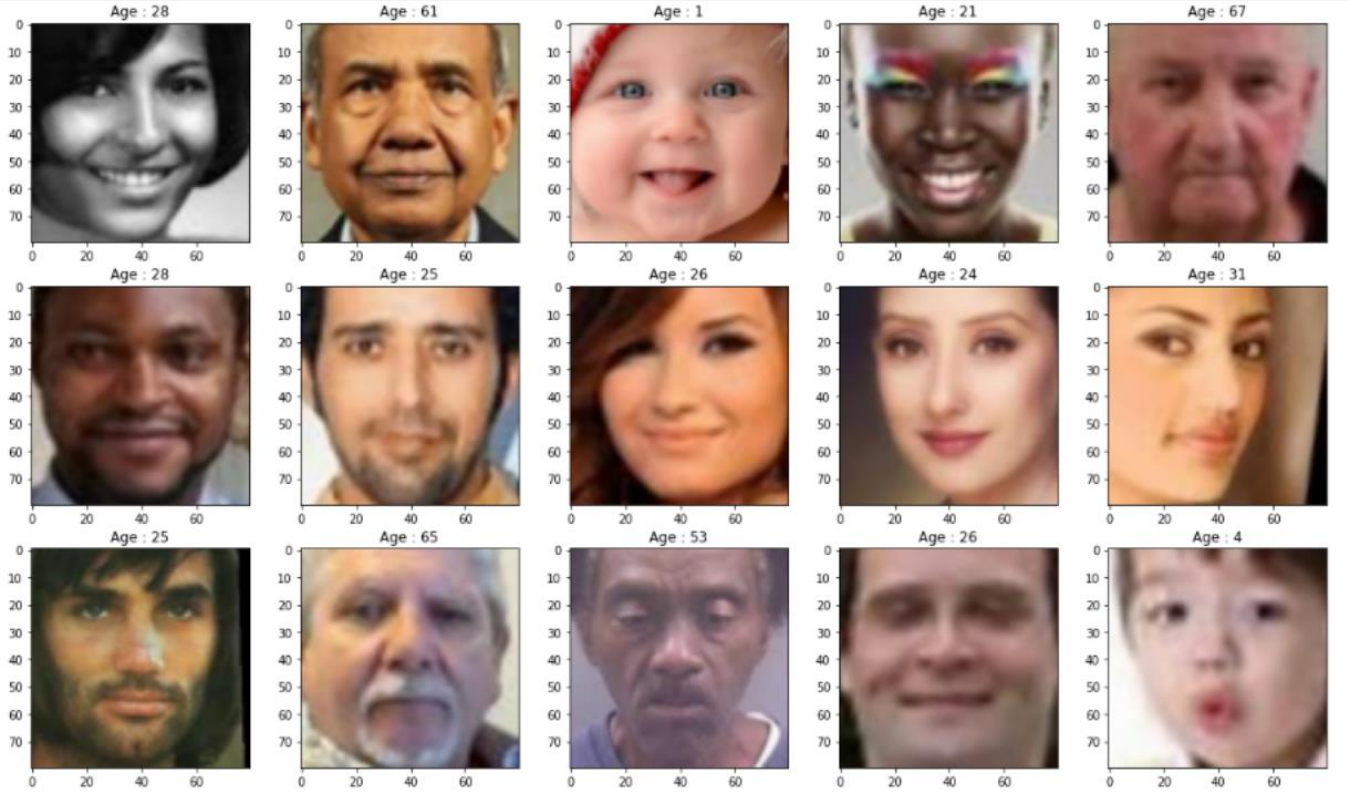
$$\hat{\mathbf{w}}_{\mathrm{ML}} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma^{2}}}\right) + \frac{\left(y_i - \mu_{x_i}\right)^2}{2\sigma^2}$$

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

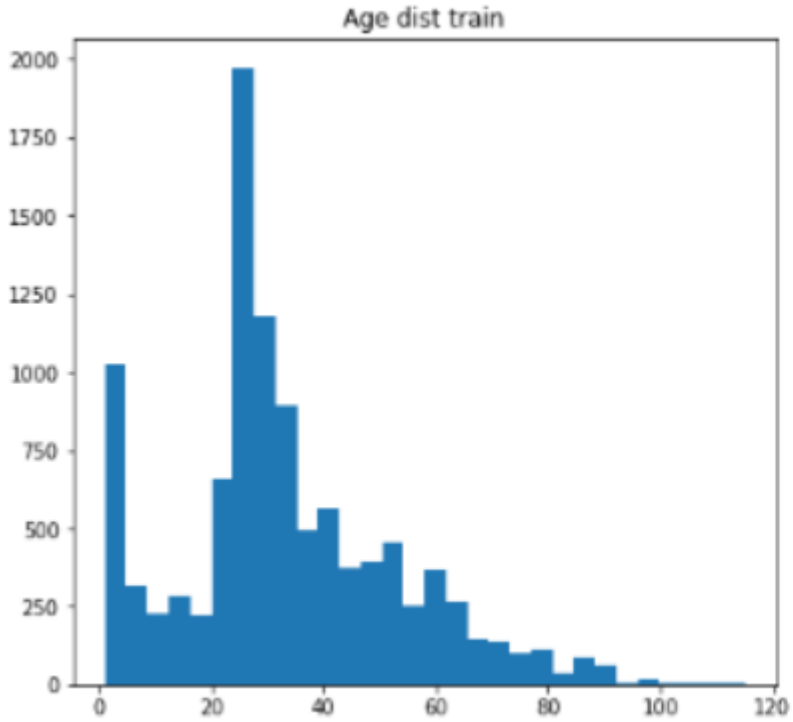
Exercise: bis um 16:00

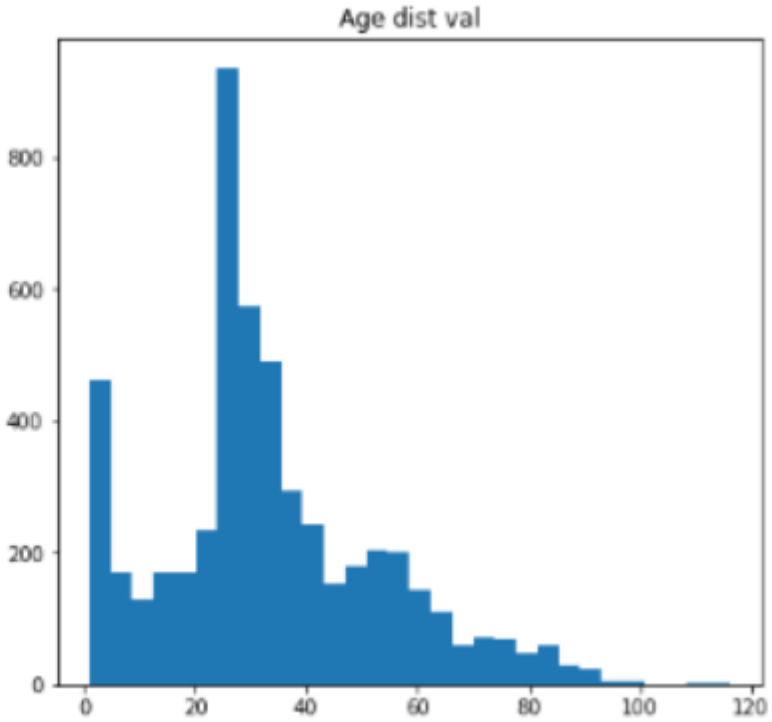


Check out the models for age prediction with flexible and constant variance and try to understand the code and to answer the questions.

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

Age distribution

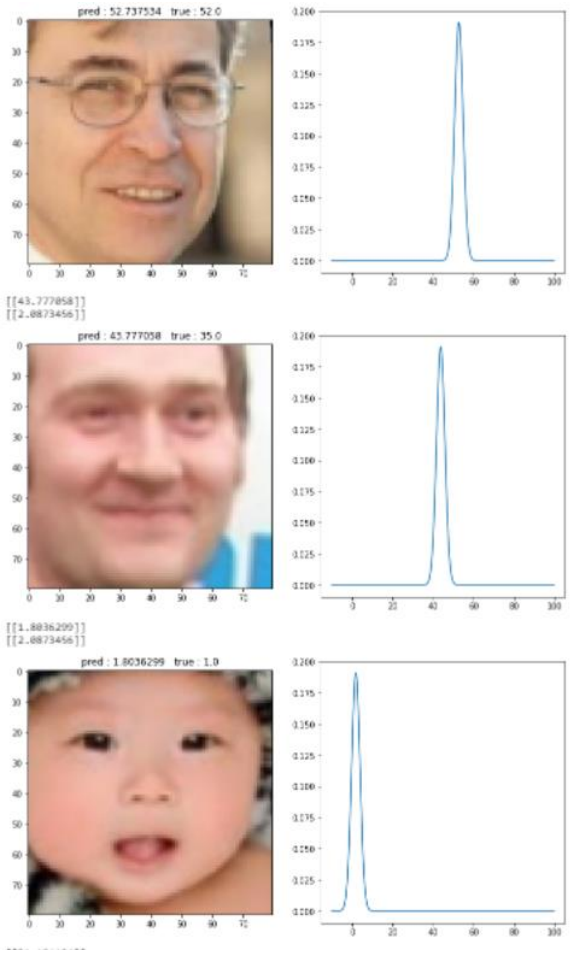




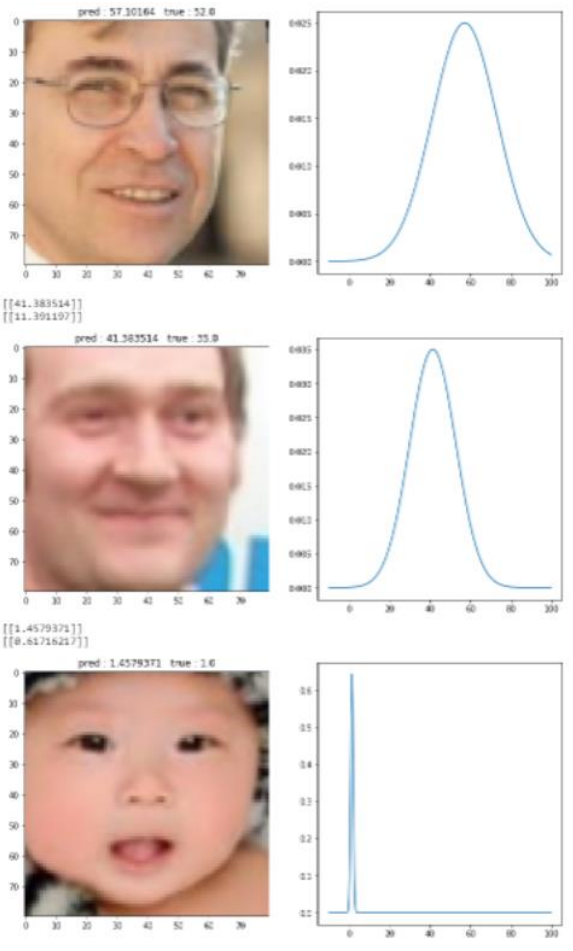
We have a lot of small children in the data set, for whom the age estimation is probably not so difficult.

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.

Where does a CNN look at?

Visualize patches yielding high values in activation maps

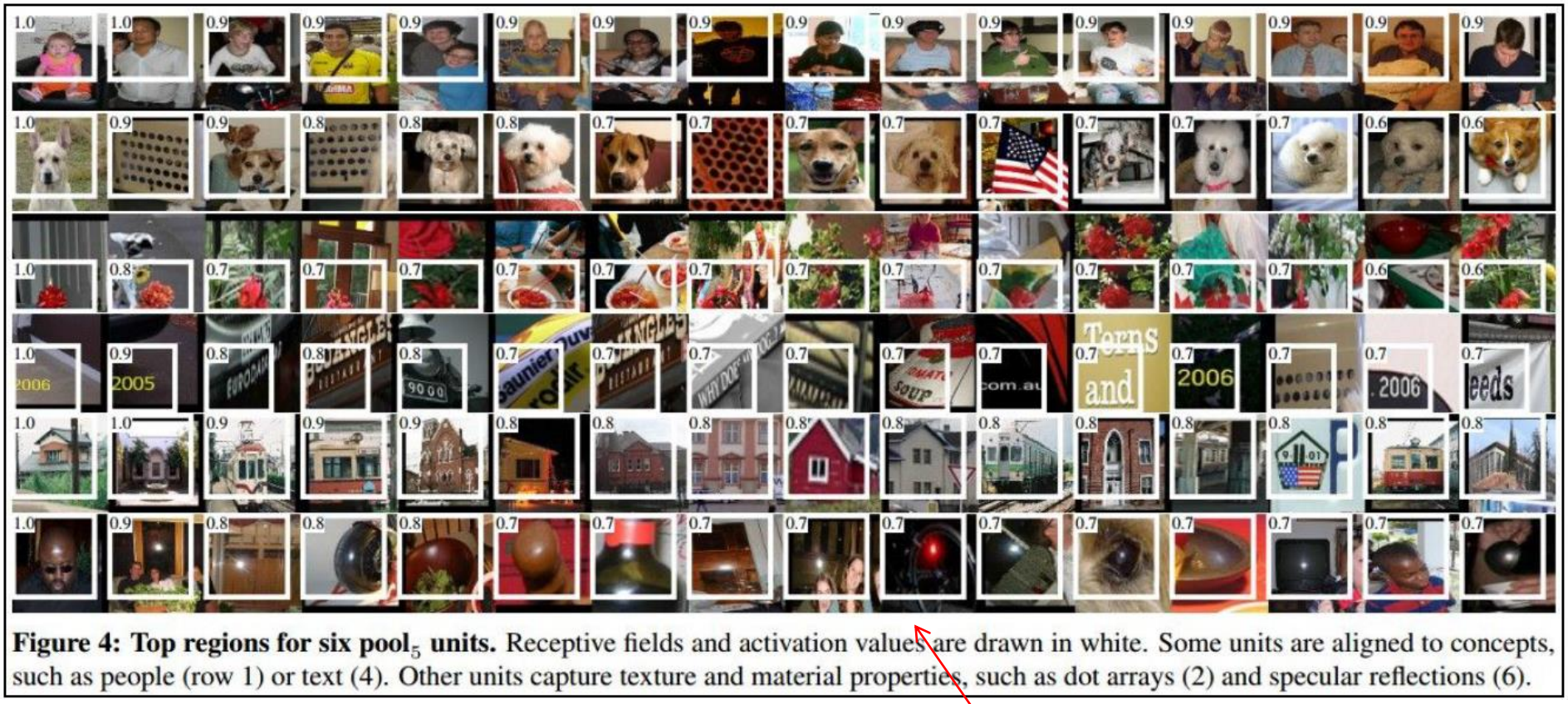
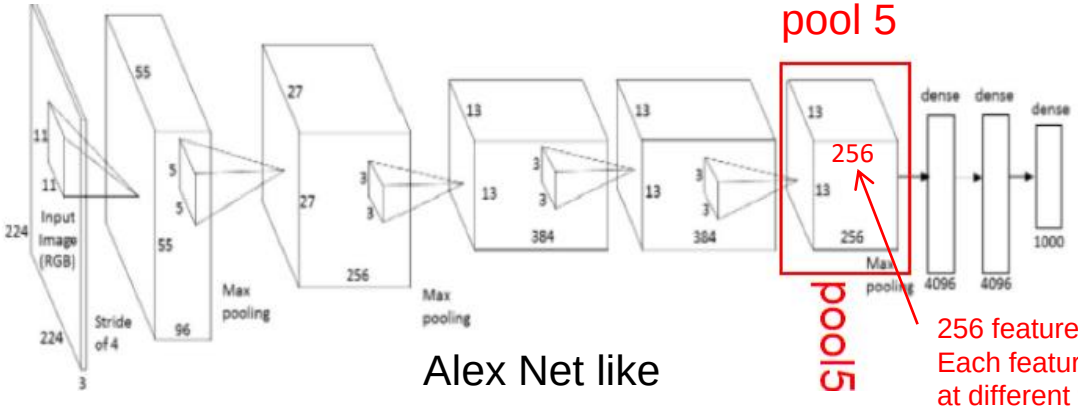


Figure 4: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).



http://cs231n.github.io/understanding-cnn/

Here we show image patches that activate maps in layer 5 most.

256 feature maps generated by 256 different 3x3 filters. Each feature map consists of equivalent neurons looking at different positions of the input volume.

What kind of image (patches) excites a certain neuron corresponding to a large activation in a feature map?

10 images from data set leading to high signals 6 feature maps of **conv6**

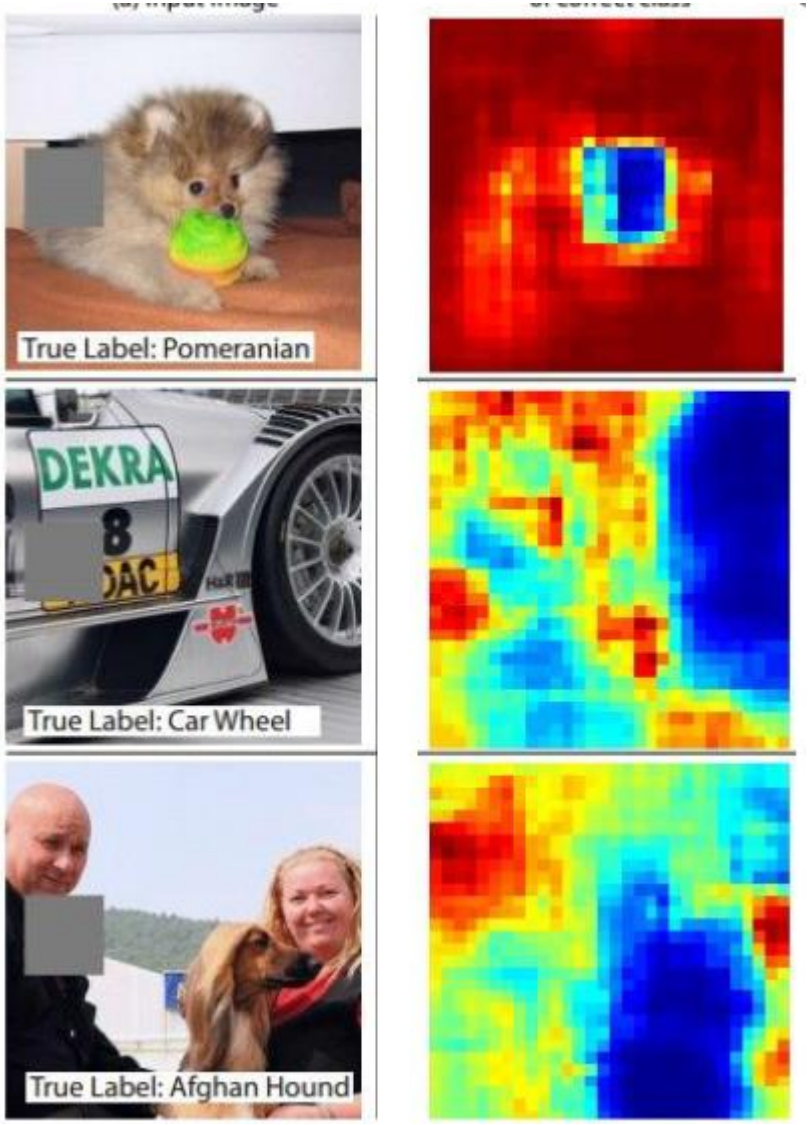
10 images from data set leading to high signals 6 feature maps of **conv9**

image credit: cs231n

Which pixels are important for the classification?
Occlusion experiments

Occlude part of the image with a mask
and check for each position of the mask how strongly the score for the correct class is changing.

Warning:
Usefulness depends on application…



Occlusion experiments [Zeiler & Fergus 2013]

image credit: cs231n

Which pixels are important for the classification?
LIME: Local Interpretable Model-agnostic Explanations

Idea:

1) perturb interpretable features of the instance – e.g. randomly delete super-pixels in an image and track as perturbation vector such as (0,1,1,0,…,1)=x.
2) Classify perturbed instance by your model, here a CNN, and track the achieved classification-score=y
3) Identify for which features/super-pixels the presence in the perturbed input version are important to get a high classification score (use RF or lasso for y~x)

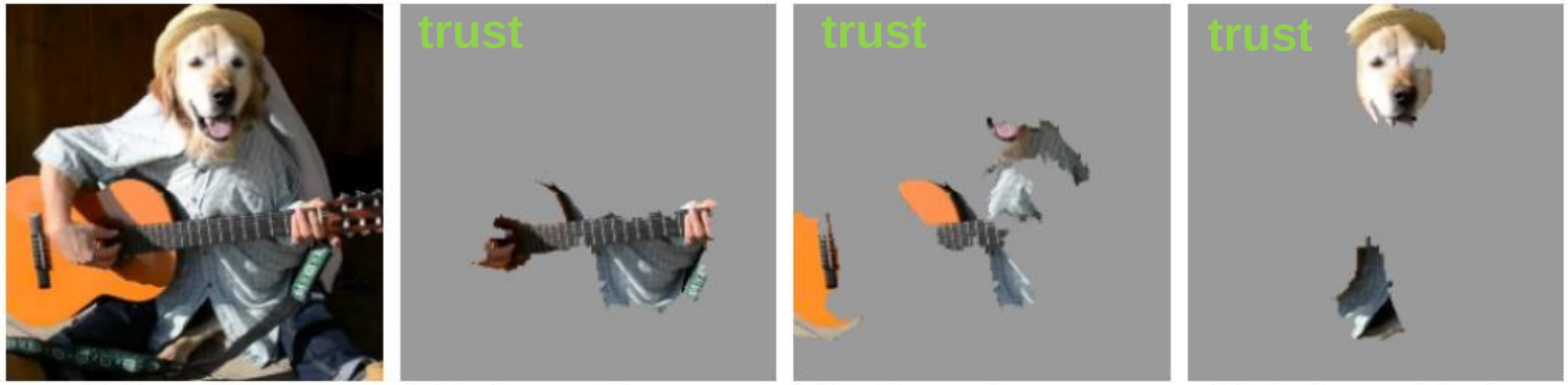
https://arxiv.org/abs/1602.04938

(a) Husky classified as wolf (b) Explanation

-> presence of snow was used to distinguish wolf and husky

-> Explain the CNN classification by showing instance-specific important features

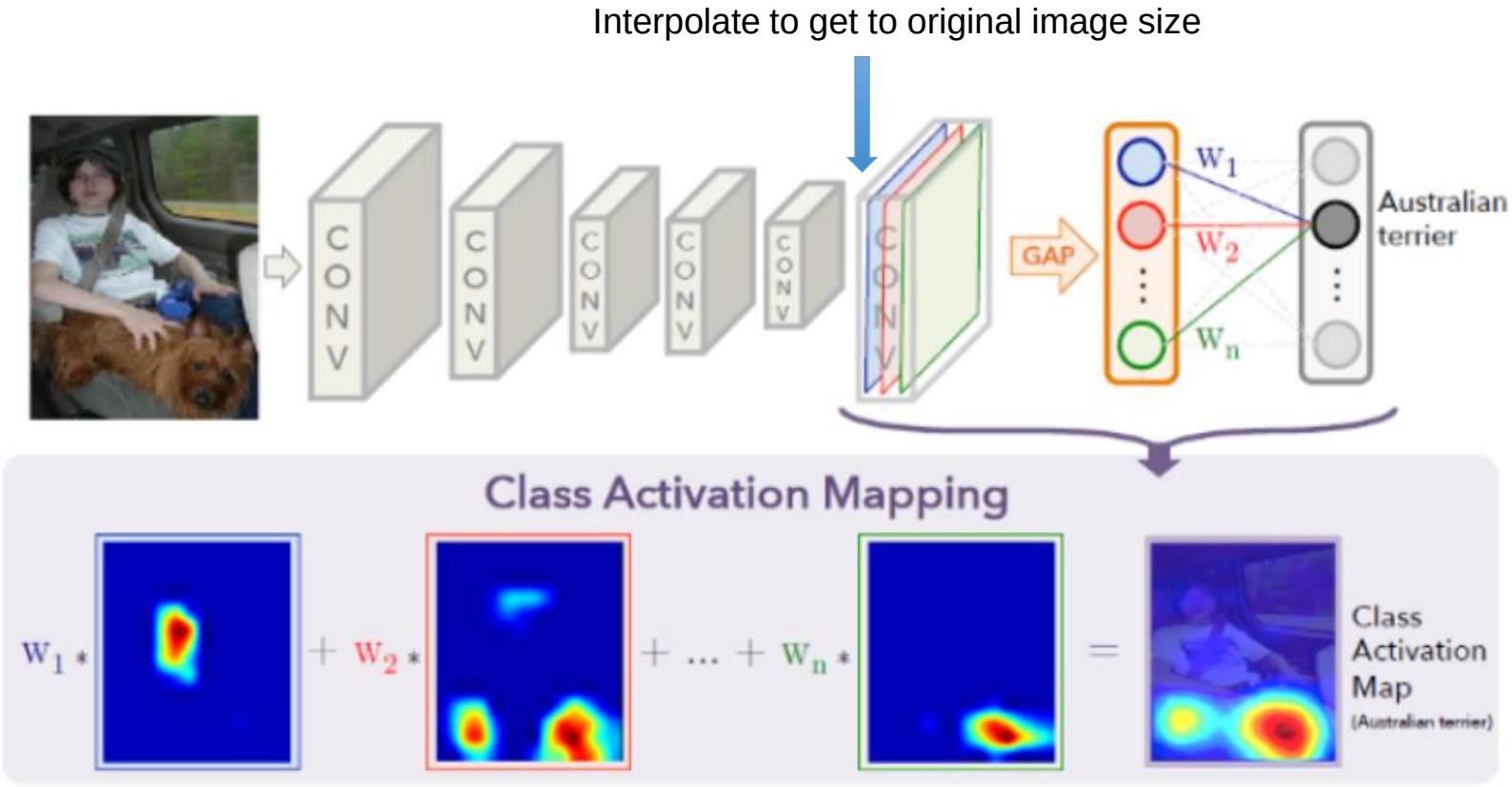
visualize important feature allows to judge the individual classification



(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Tf, python code: https://github.com/marcotcr/lime -> image tutorial
https://github.com/marcotcr/lime/blob/master/doc/notebooks/Tutorial%20-%20Image%20Classification%20Keras.ipynb

Class Activation Mapping (CAM)

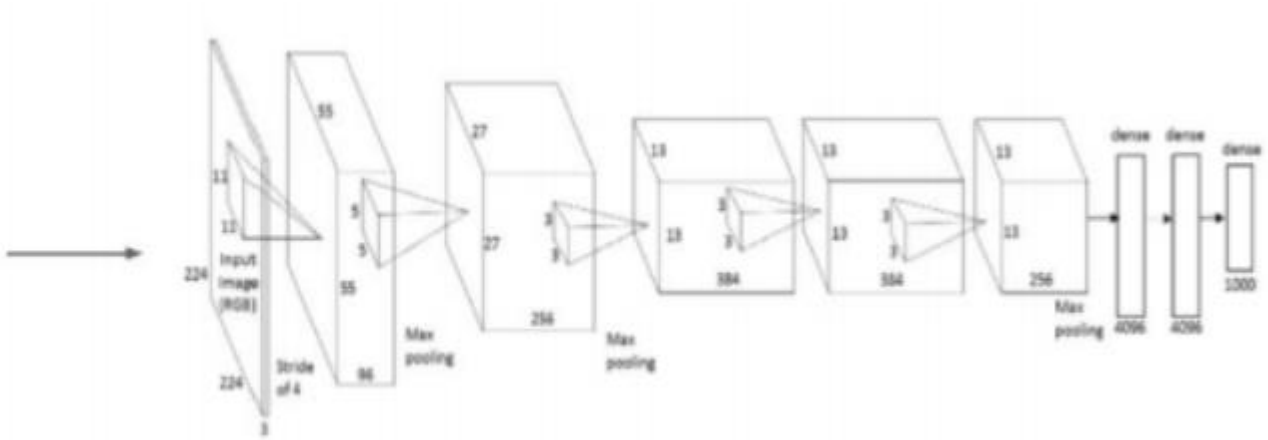


We compute the CAM by multiplying each feature map of the final convolution layer with the corresponding weight connected to the neuron of the winning class.

https://arxiv.org/abs/1512.04150

https://towardsdatascience.com/class-activation-mapping-using-transfer-learning-of-resnet50-e8ca7cfd657e

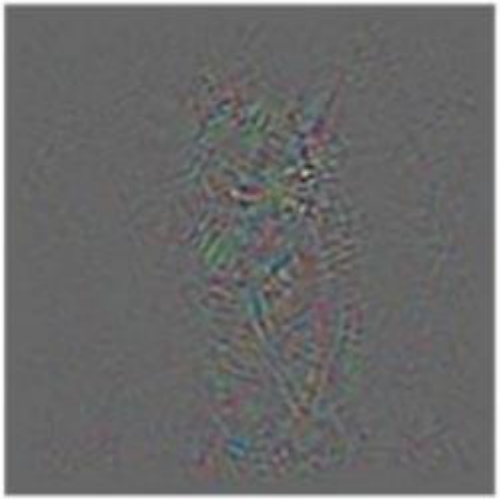
Gradient Backpropagation



1) Do a forward pass with the image

2) Compute the gradient of the winning class neuron using backprop

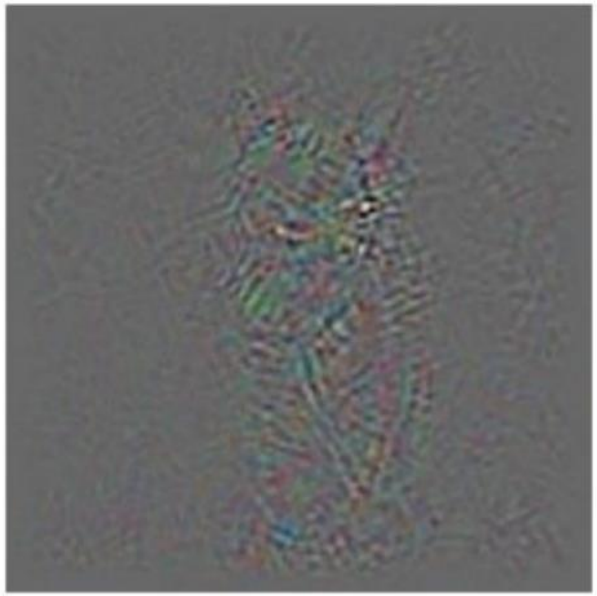
   Which pixels would make the class neuron not turn on if they had been different?
   In other words, for which inputs is $\partial(\text{class} - \text{neuron})/\partial xi$ large?



https://www.cs.toronto.edu/~guerzhoy/321/lec/W07/HowConvNetsSee.pdf

Guided Gradient Backpropagation

- We are only interested to see which image features the neuron detects, not in what kind of stuff it doesn't detect
- So only propagating positive gradients (set all the negative gradients to 0)



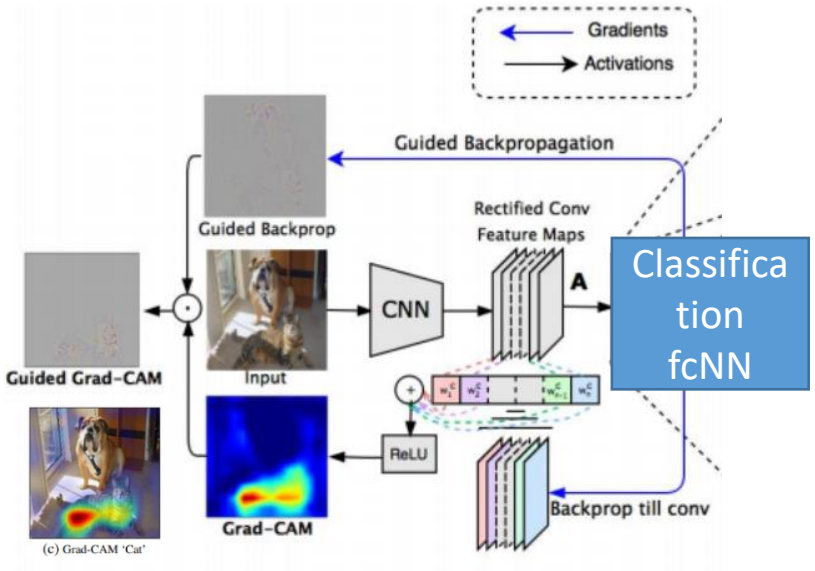
Backprop

Guided Backprop

https://www.cs.toronto.edu/~guerzhoy/321/lec/W07/HowConvNetsSee.pdf
Code: see iNNvestigate colab NB https://github.com/avciidp/explainable_ai/blob/master/iNNvestigate_fashion_light.ipynb
paper: https://arxiv.org/abs/1808.04260

Grad-Cam: Gradient based Class Activation maps

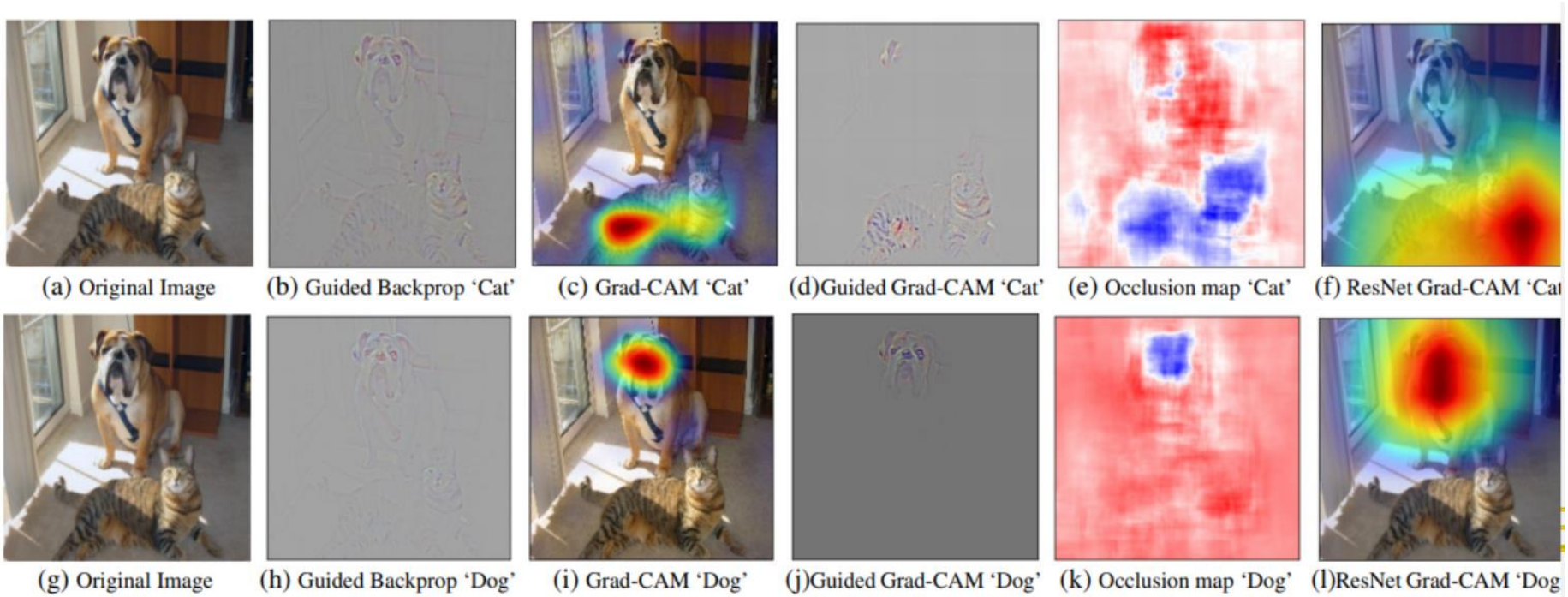
Grad-Cam: Use Guided Backprop to scale contributions in CAMs



- Set gradient to zero for all classes expect the predicted class
- Backpropagate this gradient to the last conv-layer
  -> get coarse grad-CAM localization
- Multiply coarse grad-CAM with the guided back-prop signal

https://arxiv.org/abs/1610.02391
Code: https://keras.io/examples/vision/grad_cam/
https://www.pyimagesearch.com/2020/03/09/grad-cam-visualize-class-activation-maps-with-keras-tensorflow-and-deep-learning/

What in the image was important for the prediction?



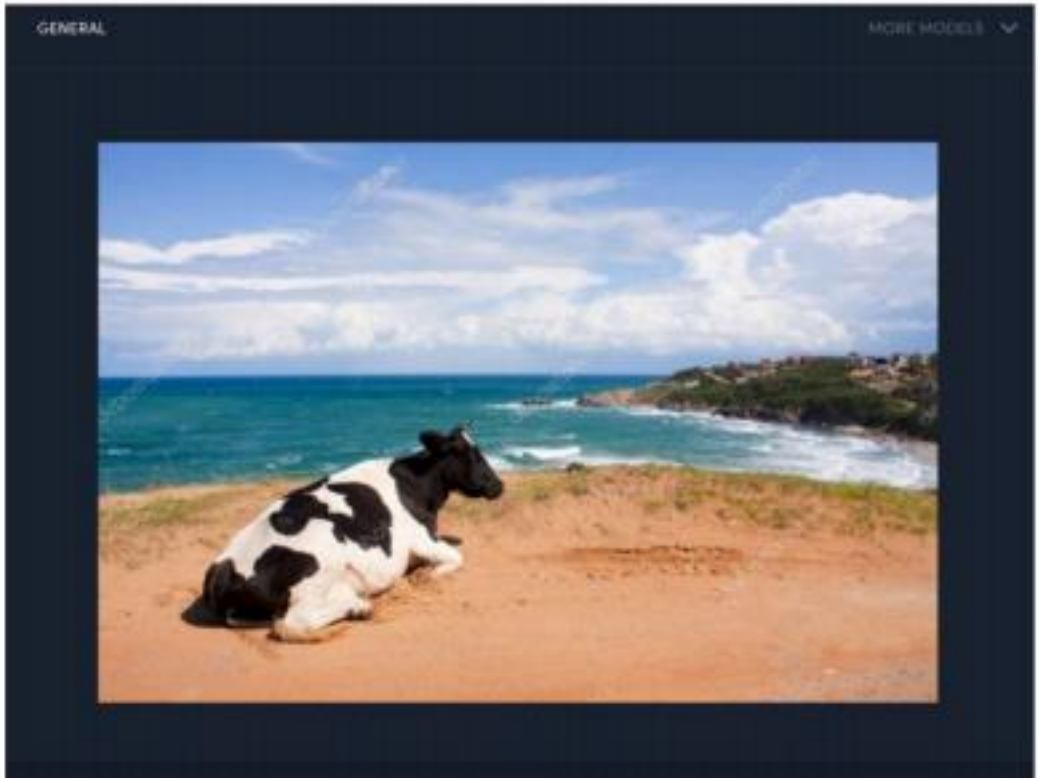
Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization https://arxiv.org/abs/1610.02391

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

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

A high performant NN
(trained on imageNet data)
does not see the elephant!

General VIEW DOCS

PREDICTED CONCEPT	PROBABILITY
group	0.979
adult	0.977
people	0.976
furniture	0.960
room	0.957
business	0.903
indoors	0.901
man	0.896
seat	0.895

NeurIPS 2018 Workshop on Causal Learning

Elephant in the room

- Aufgabe: https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/18_elephant_in_the_room.ipynb

If time allows:

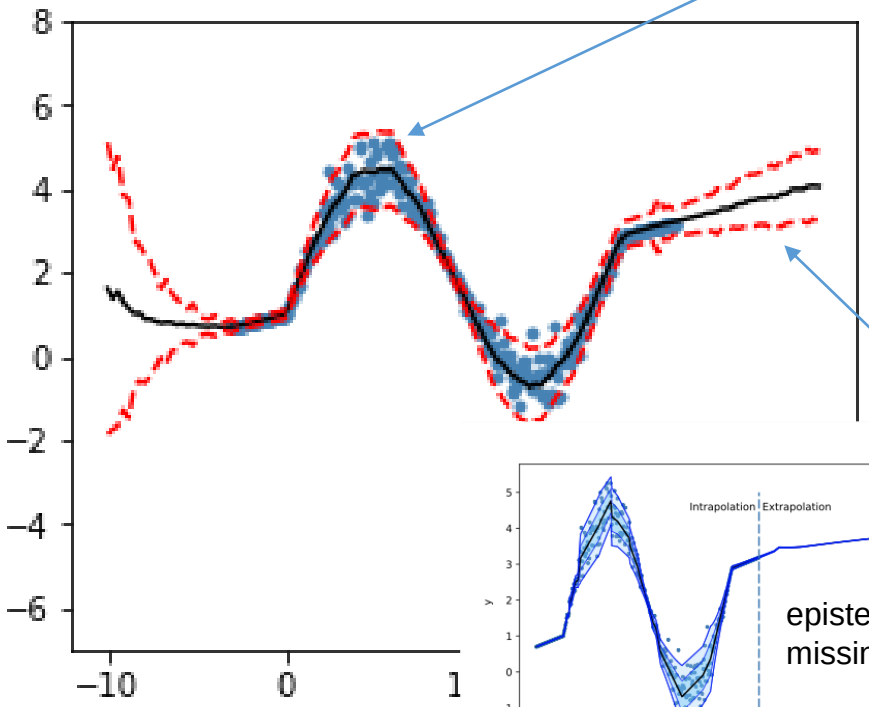
- Issues with current DL approach
  - No uncertainty for the fitted weights
    → if algorithmic and epistemic uncertainty is ignored :
    - No increased uncertainty in case of extrapolation
    - Deficits in prediction performance

- Approaches to take algorithmic and epistemic uncertainty into account:
  - Deep Ensembling
  - MC Dropout

Aleatroic vs. Epistemic Uncertainty

Much spread in train data → aleatroic uncertainty (from latin "Alea Acta est")

Caesar: Die Würfel sind gefallen!



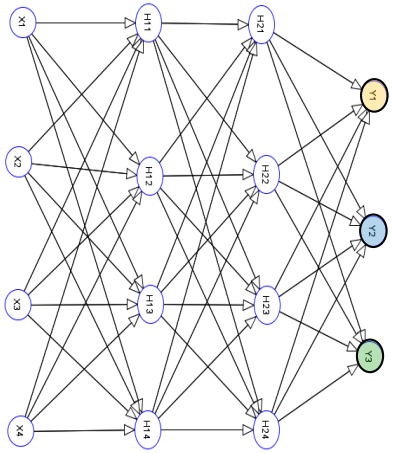
We understand a dice perfectly (no epistemic uncertainty), but still there is a aleatoric uncertainty about the the number that will show up next when rolling the dice.

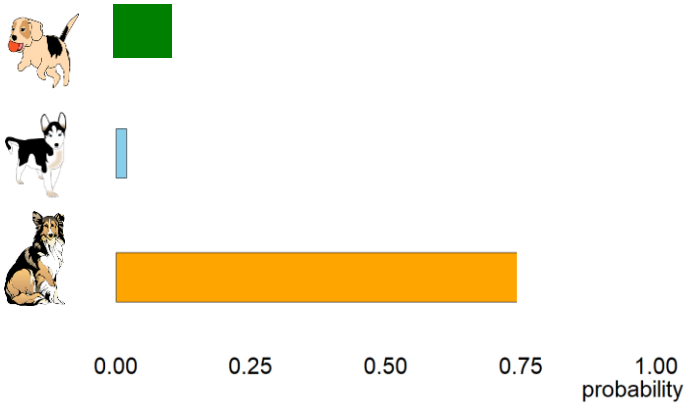
epistemic uncertainty is missing in classical NN

No (or few) train data
→ no (or few) "knowledge"
(from Ancient Greek ἐπιστήμη *(epistḗmē)* 'knowledge')
→ epistemic uncertainty

- *Aleatoric* uncertainty is due to the uncertainty, that is inherent in the data.
- The uncertainty when leaving the 'known ground' is called *epistemic* uncertainty.

CNNs have high performance on in-distribution examples

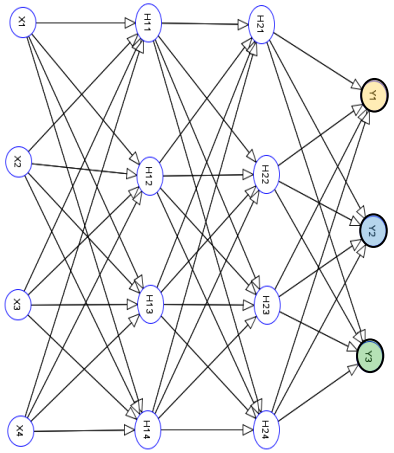




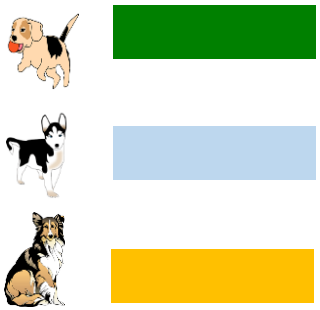
CNNs yield high accuracy and calibrated probabilities, but…

A classical NN cannot ring the alarm in case of out-of-distribution (OOD) examples

What happens if we present a novel class to the CNN?

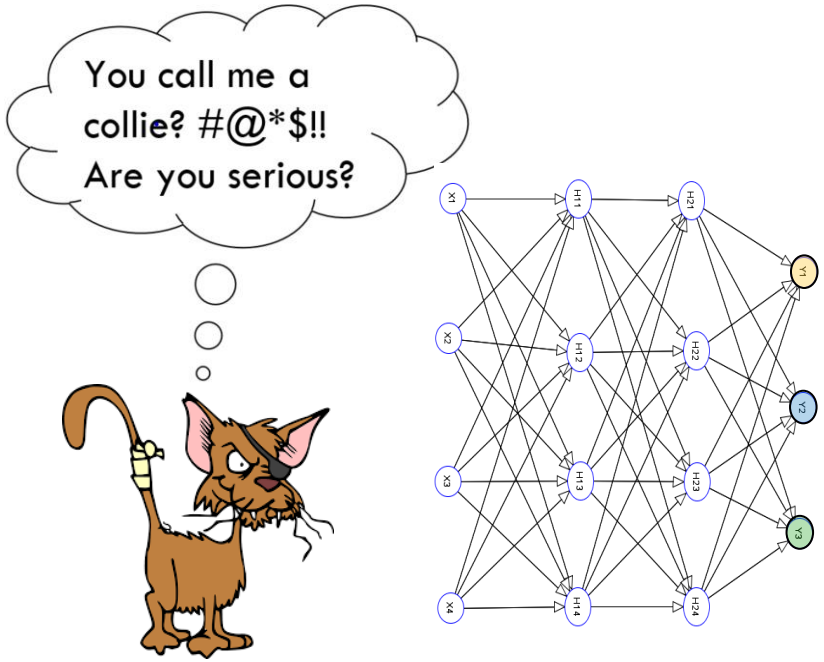


You might expect:



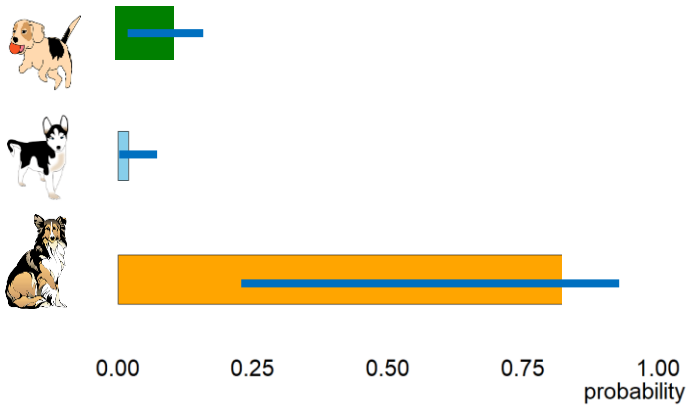
A classical NN cannot ring the alarm in case of out-of-distribution (OOD) examples

What happens if we present a novel class to the CNN?



But you probably get:

Plain wrong !



We need some error bars!
We need epistemic uncertainty!

Importance to detect OOD (out of distribution)

- Current DL Systems bad in out of distribution OOD situations
- Application need at least to detect OOD situations

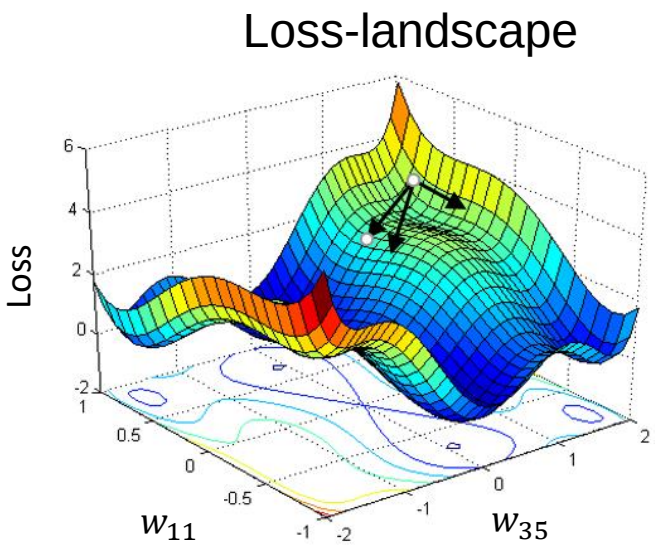
Foto: https://www.crn.de/software-services/halloween-training-fuer-google-autos.108409.html

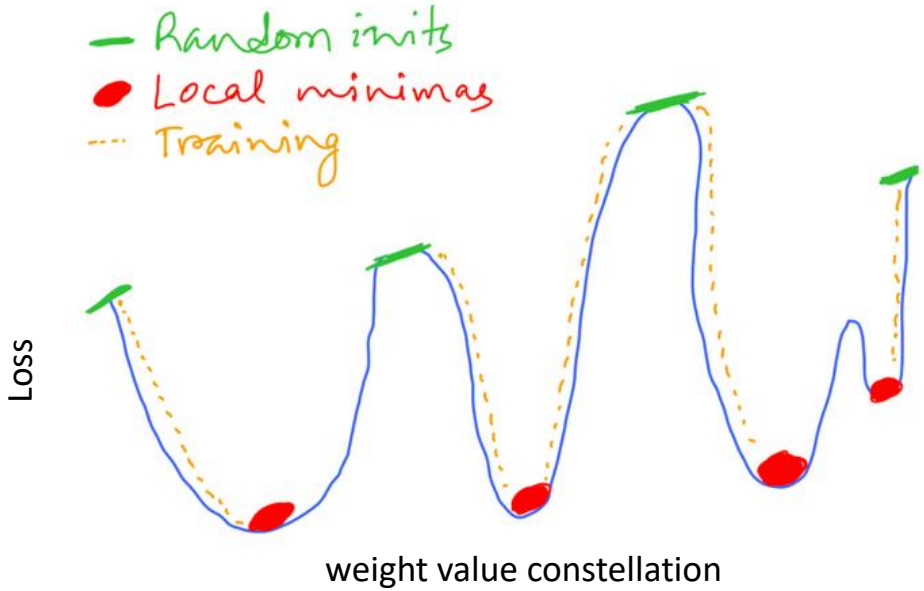
Deep ensembles

→ addressing algorithmic uncertainty

https://www.tutorialspoint.com/how-can-keras-be-used-to-implement-ensembling-in-python

The loss-landscape in DL is usually not convex



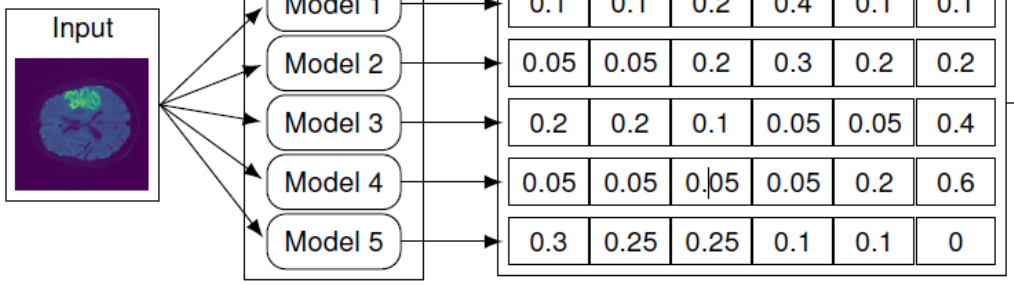


The loss-landscape of DL models has many local minima with similar depth.

Training is started with a random weight value initialization → training the NN with SGD and the same data several times is usually ending in different local minima.

Image credit: https://wandb.ai/authors/loss-landscape/reports/Understanding-the-Effectivity-of-Ensembles-in-Deep-Learning--VmlldzoxODAxNjA

Deep ensembling:
Train several NN models and average their predicitions

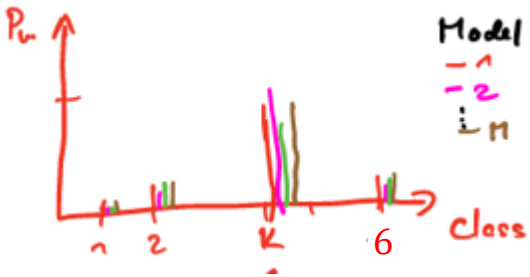


Each model predictis probabilities for the 6 classes

p_1	p_2	p_3	p_4	p_5	p_6
0.1	0.1	0.2	0.4	0.1	0.1
0.05	0.05	0.2	0.3	0.2	0.2
0.2	0.2	0.1	0.05	0.05	0.4
0.05	0.05	0.05	0.05	0.2	0.6
0.3	0.25	0.25	0.1	0.1	0

We get the ensemble predictions by averaging the probabilities that were predicted by the different models for each class

p_1^E	p_2^E	p_3^E	p_4^E	p_5^E	p_6^E
0.14	0.13	0.16	0.18	0.13	0.26



We can get the aleatoric ensemble uncertainties by computing the entropy of the ensemble prediction $H(P^E) = -\sum_i p_i^E \cdot \log(p_i^E)$.
We can get the epistemic ensemble uncertainties by e.g. computing the standard deviation of the probabilities that were predicted by the different models for each class

sd_1^E sd_2^E sd_3^E sd_4^E sd_5^E sd_6^E

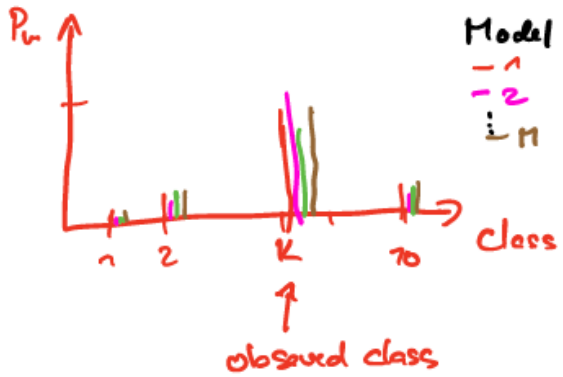
Nice:
For the convex NLL loss, it is guaranteed, that the NLL of the ensemble prediction is better (smaller or equal) than the average NLL of the individual models.

Original paper by Lakshminarayanan et al.: https://arxiv.org/pdf/1612.01474.pdf

Ensembling improves the NLL performance

Ensemble prediction for an observation with observed class = k, based on M models:

$$p_k^E = \frac{1}{M} \sum_{m=1}^{M} p_k \qquad k = 1 \dots 10 \text{ for 10 classes}$$



associated NLL contribution ℓ:

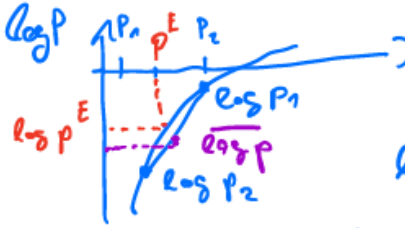
Model m: $\ell_m = -\log p_{km}$; Ensemble: $\ell^E = -\log p_k^E$

to show $\bar{\ell} \geq \ell^E$ with $\bar{\ell} = \frac{1}{M} \sum_{m=1}^{M} \ell_m = \frac{1}{M} \sum_m -\log p_{km}$

$$= -\underbrace{\frac{1}{M} \sum_m \log p_{km}}_{\leq \log \frac{1}{M} \sum_m p_{km}}$$

$$\geq -\log p_k^E = \ell^E \qquad \square$$

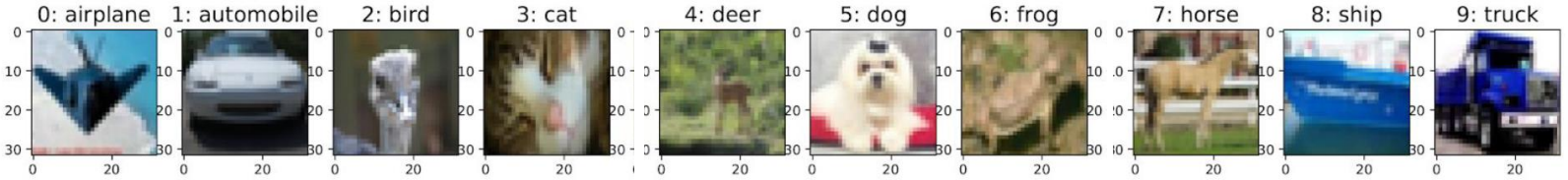
use Jensen inequality

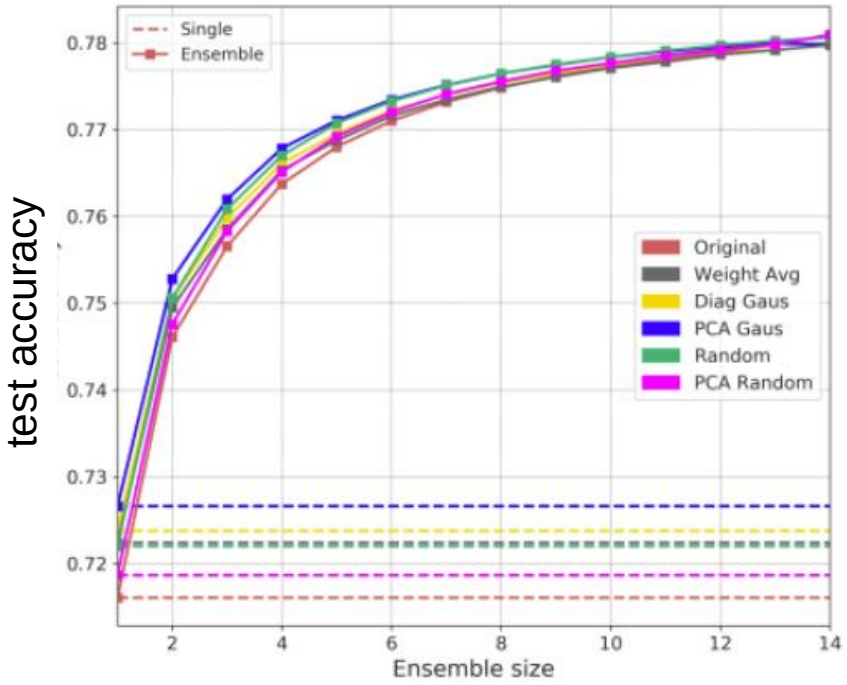


$$\log \bar{p} \geq \overline{\log p}$$

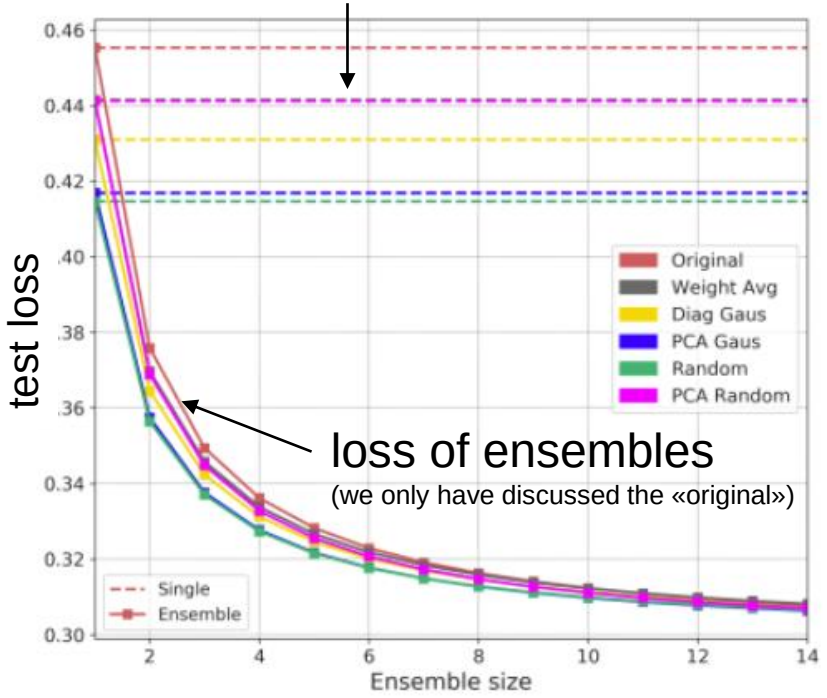
$$\log \frac{1}{M} \sum_m p_m \geq \frac{1}{M} \sum_m \log p_m$$

Deep ensembling improves prediction power





loss of 5 single models



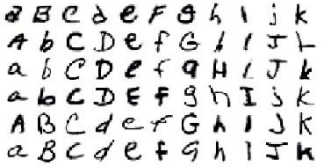
Ensembes with as few as 3 or 5 members are typically enough to achieve a perfromance gain.

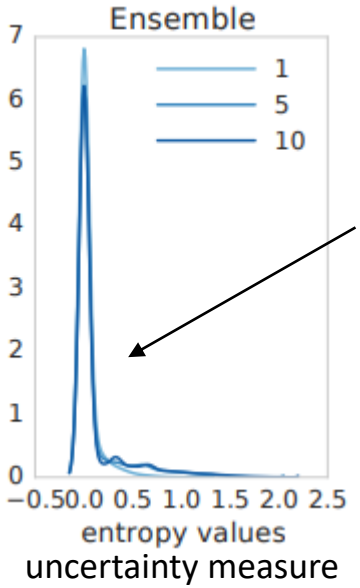
Image credit: https://wandb.ai/authors/loss-landscape/reports/Understanding-the-Effectivity-of-Ensembles-in-Deep-Learning--VmlldzoxODAxNjA

Deep ensembles improve uncertainty measures

We want that a model, that is trained on normal MNIST letter data, should provide large uncertainties when applied on novel (NOT-MNIST) letter images.

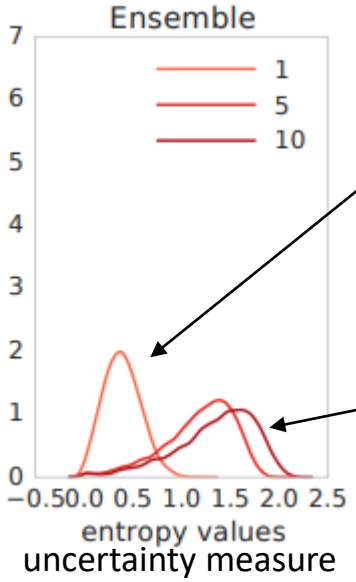
Uncertainties for prediction of **normal MNIST** letter images





Ensemble uncertainty for a known class stays small for all sizes of the ensemble

Uncertainties for prediction of **NOT-MNIST** images



A single models shows less uncertainty for a novel class than an ensemble model

Ensemble uncertainty measures for novel classes improve with #members per ensembles

Image credits Lakshminarayanan et al.: https://arxiv.org/pdf/1612.01474.pdf