PROGRAMMING IN PYTHON II

Making more of Data: Data Augmentation

Michael Widrich
Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

Outline

1. Motivation
2. Data augmentation
3. Using all the data
4. Important: Communication of used data
5. PyTorch
6. Scenario: Self-driving cars
7. Python II Project

Motivation

- NN models with high complexity can solve complex tasks
- Higher complexity of model leads to quicker over-fitting on training data
- We require a lot of data to not over-fit and achieve good generalization
- Problem: Collecting data is expensive
- Solution: Creative usage/augmentation of data

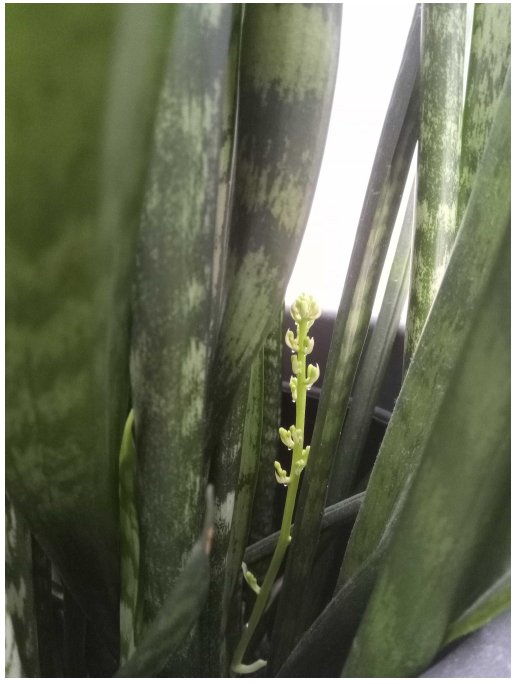
Data augmentation

- In many cases, we can augment data artificially without collecting new real samples
  - Create “new” artificial training samples by modifying existing samples
  - Only use augmented data for training, not for evaluation (we want to estimate generalization on real data!)
- Pros:
  - Can increase the number of data points by a large factor with little effort (often on-the-fly)
  - Reduces over-fitting, increases robustness of model
- Cons:
  - Can introduce artifacts or change the task
  - Heavily dependent on data, model, and task

Noise

- Idea: Add random additive noise to input features, such that input feature values are still plausible
  - Simple to implement
  - Model will be more robust w.r.t. input noise (e.g. sensor noise)
  - Usable for many types of input data (dense features, image data, timeseries, . . .)
  - Noise strength is a hyperparameter
- Pitfalls:
  - Noise might introduce change of input distribution
    - Usually zero mean and normal or uniform distributed noise but depends on input distribution
  - For binary features it might help to have binary noise

Noise: Example



original

noise

$\mathcal{N}(\mu = 0, \sigma = 0.1)$

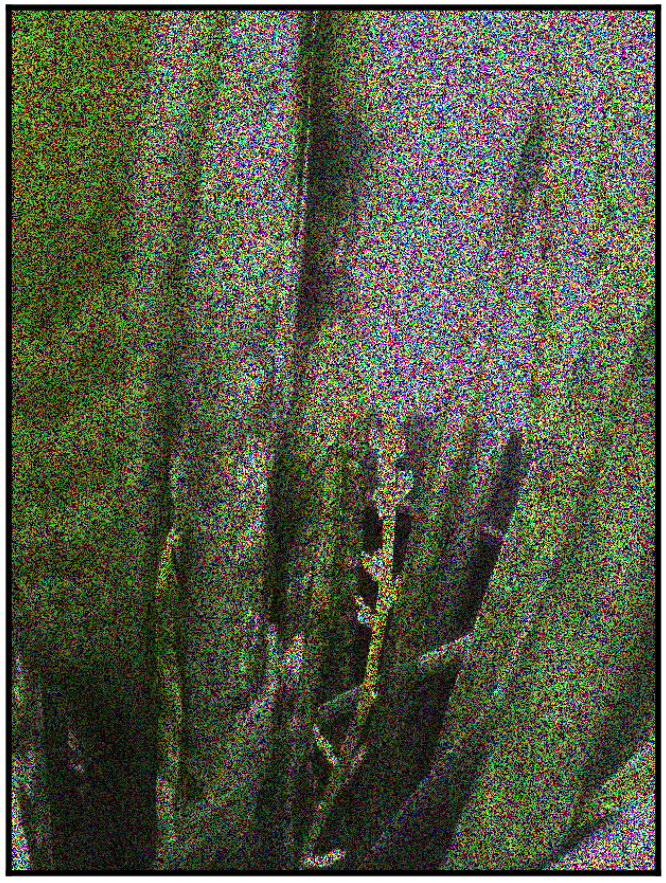
noise

$\mathcal{N}(\mu = 0, \sigma = 0.5)$

Input Dropout

- Idea: Temporarily remove input features, such that model has to consider more features for prediction
  - Dropping out random feature values, whole features/channels/timesteps, or groups of values (e.g. adjacent pixels, see “DropBlock” and “Random Erasing Data Augmentation”)
  - Image data: Highly correlated adjacent pixels require dropout of areas of adjacent pixels (redundant information)
  - Dropped-out values typically set to 0 but depends on drop-out version
  - Probability of dropping out values is a hyperparameter
  - Compensation for shift in distribution might be necessary
    - Automatically done by `torch.nn.Dropout`, `torch.nn.AlphaDropout`, etc.

Input Dropout: Example



dropout random pixels

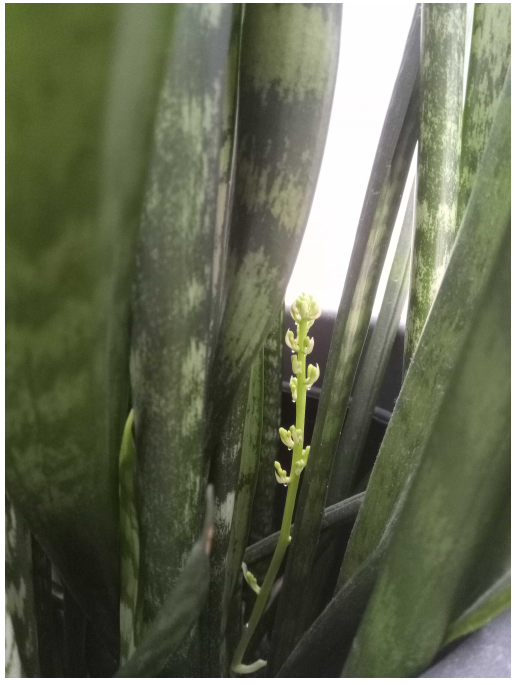
adjacent pixels over all channels

Image-based data

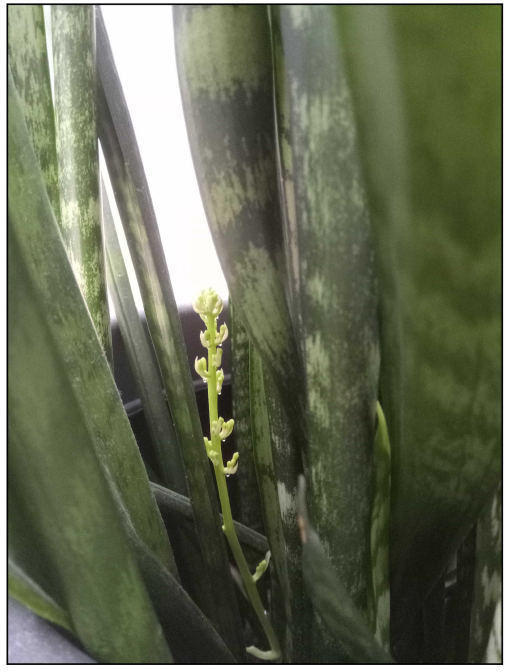
- Data augmentation is dependent on the data
- Data with special properties (e.g. spatial or temporal data) allows for additional augmentation methods
- The following slides show augmentation methods for image data (=spatial data)
- Warning: Always check if the augmentation method makes sense for the task/data!

Image-based data: Flipping

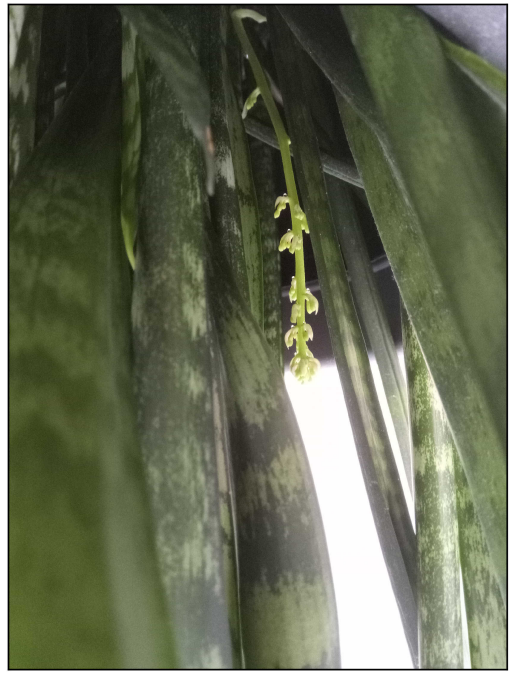
- Flipping: Mirror image vertically or horizontally



original



horizontal flip



vertical flip

Image-based data: Crop/Zoom

- Crop/Zoom: Select part of image as new image, optionally zoom in or out

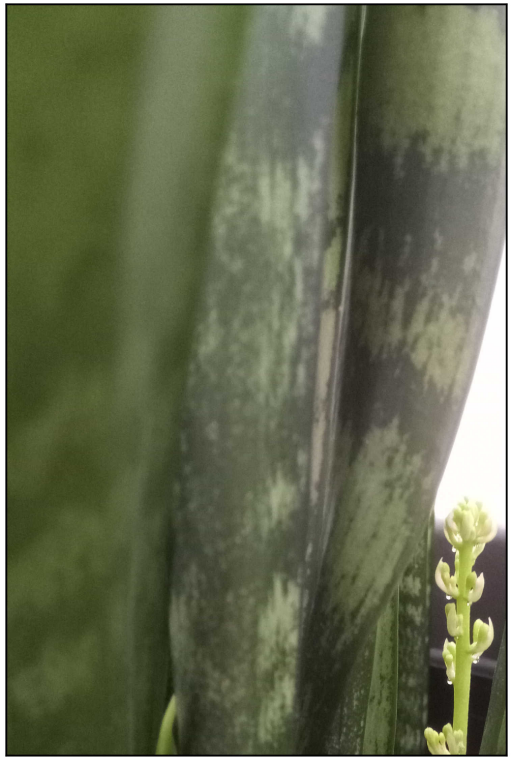


Image-based data: Rotation

- Rotation: Rotate image in various angles

Image-based data: Distortion effects

- Distortion effects: Warp image using e.g. lens-distortion-like effects

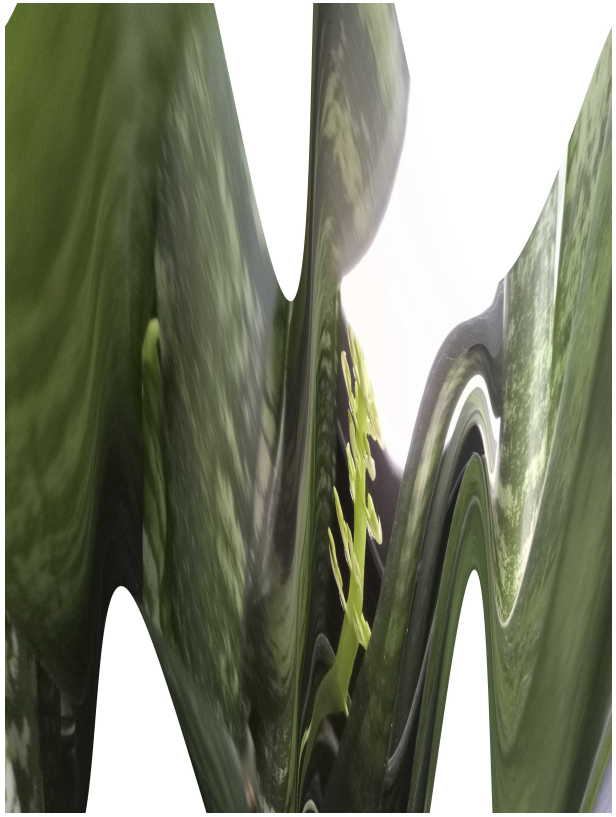
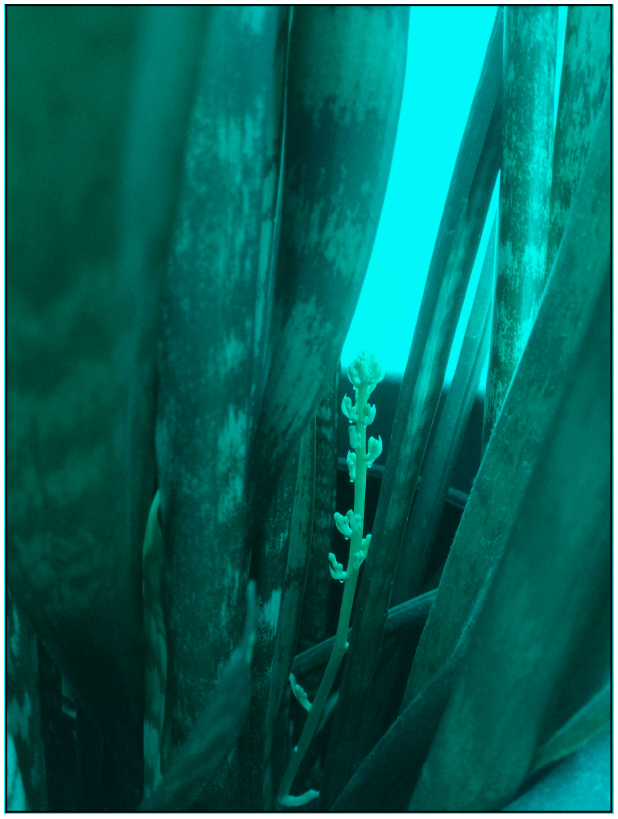


Image-based data: Color augmentation

- Color jittering: Modify color channels, brightness, hue, saturation, or value



Using all the data (1)

- Data augmentation is not the only method to increase the size of our dataset
- Using auxiliary tasks
  - Often training data comes with metadata - use this data as auxiliary targets
  - Introduces additional information to model via the gradient
  - Allows for usage of samples without information on the main target
- Add semi-supervised task (e.g. as auxiliary targets)
  - Can you create auxiliary targets from the data?
  - Classic: Next-frame or next-timestep prediction

Using all the data (2)

- Pre-training the model on external data
  - Is there a larger dataset that is similar to yours/relevant for you task?
  - Pretrain model on that data, then fine-tune on your dataset
  - You can pre-train on auxiliary targets if your main target is not given
- Use meta-learning (this might be the standard in the future?)
  - Still under research but quite promising
  - Train a meta-learning model to train models on other data/tasks (learning-to-learn)
  - Can learn better optimizers and training strategies
  - Meta-learners require much less samples to train (useful in few-shot learning)

Important: Communication of used data

- Always communicate what data you used for training with the people who want to use the model!
- This includes the used augmentation methods, auxiliary targets, etc. and the consequences this might have on the behavior of the trained model!

Data augmentation in PyTorch

- PyTorch offers Dropout-Layers and dropout-parameters for specific modules (e.g. RNN layers)
- `torchvision` offers combinable image augmentation modules
  - Based on “transforms” (image transformation modules)
  - `https://pytorch.org/docs/stable/torchvision/transforms.html`

Scenario: Self-driving cars (1)

- Data: Video data from a camera mounted at a fixed position of a car
- Goal: Predict bus-data given as numerical value (steering motion of human driver, velocity, breaks)

[Source: `https://www.argoverse.org/`]

Scenario: Self-driving cars (2)

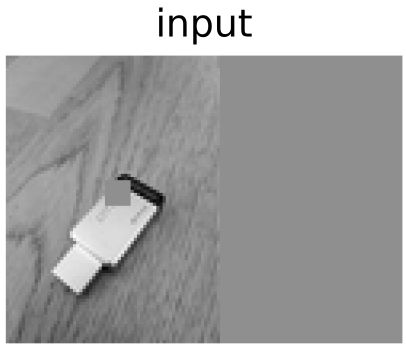
- Natural augmentation methods:
  - Noise on image (even plausible because of sensor noise)
  - Dropout of areas in input images (even plausible in case parts of camera are obscured)
- Additional data usage:
  - Next-frame prediction as auxiliary task $\rightarrow$model might learn notion of objects/physics
  - Usage of pre-trained NNs or datasets that include traffic scenes (e.g. semantic segmentation or object detection data sets/NNs)

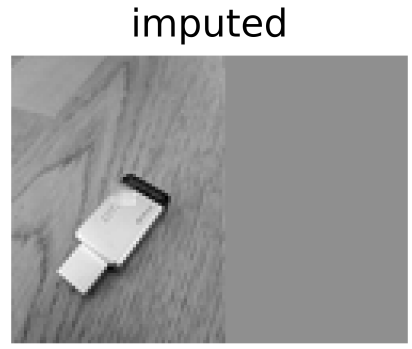
Scenario: Self-driving cars (3)

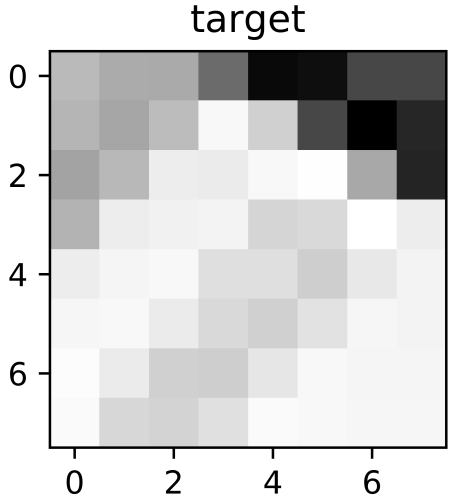
- Augmentation methods which might introduce artifacts:
  - Color augmentation (plausible with light-effects but red light might be important?)
  - Distortion-effects (maybe plausible w.r.t. light distortions on hot asphalt but use with care)
  - Rotation/Vertical flip (camera always mounted in same orientation, slopes and up/down important?)
  - Horizontal flip (text on traffic signs?)
  - Crop/zoom (camera position is fixed w.r.t. car, crop/zoom might alter view angle and perceived distance)

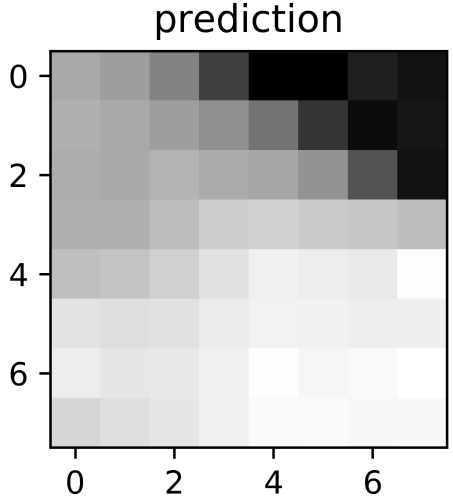
Python II Project: Data augmentation (1)

- Data: Image data, taken by cameras or scree-shots
- Goal: Predict/Restore cropped-out parts of images









Python II Project: Data augmentation (2)

- Natural augmentation methods:
  - Rotation/vertical flip/horizontal flip/crop/zoom (very plausible, camera/objects can be in different orientation or in mirrors, no real “orientation” or fixed camera position)
  - Color augmentation (plausible but target image has to be augmented in same fashion; also irrelevant since grayscale)
- Additional data usage:
  - Any image dataset (e.g. imagenet, cifar-10, cifar-100)
  - Semantic segmentation and object recognition might be useful auxiliary tasks or provide pre-trained networks

Python II Project: Data augmentation (3)

- Augmentation methods which might introduce artifacts:
  - Noise on image (plausible but noise is not predictable $\rightarrow$might train de-noiser model)
  - Dropout of areas in input images (target image has to be augmented in same fashion, might introduce artifacts of easily predicting dropped-out area shape)