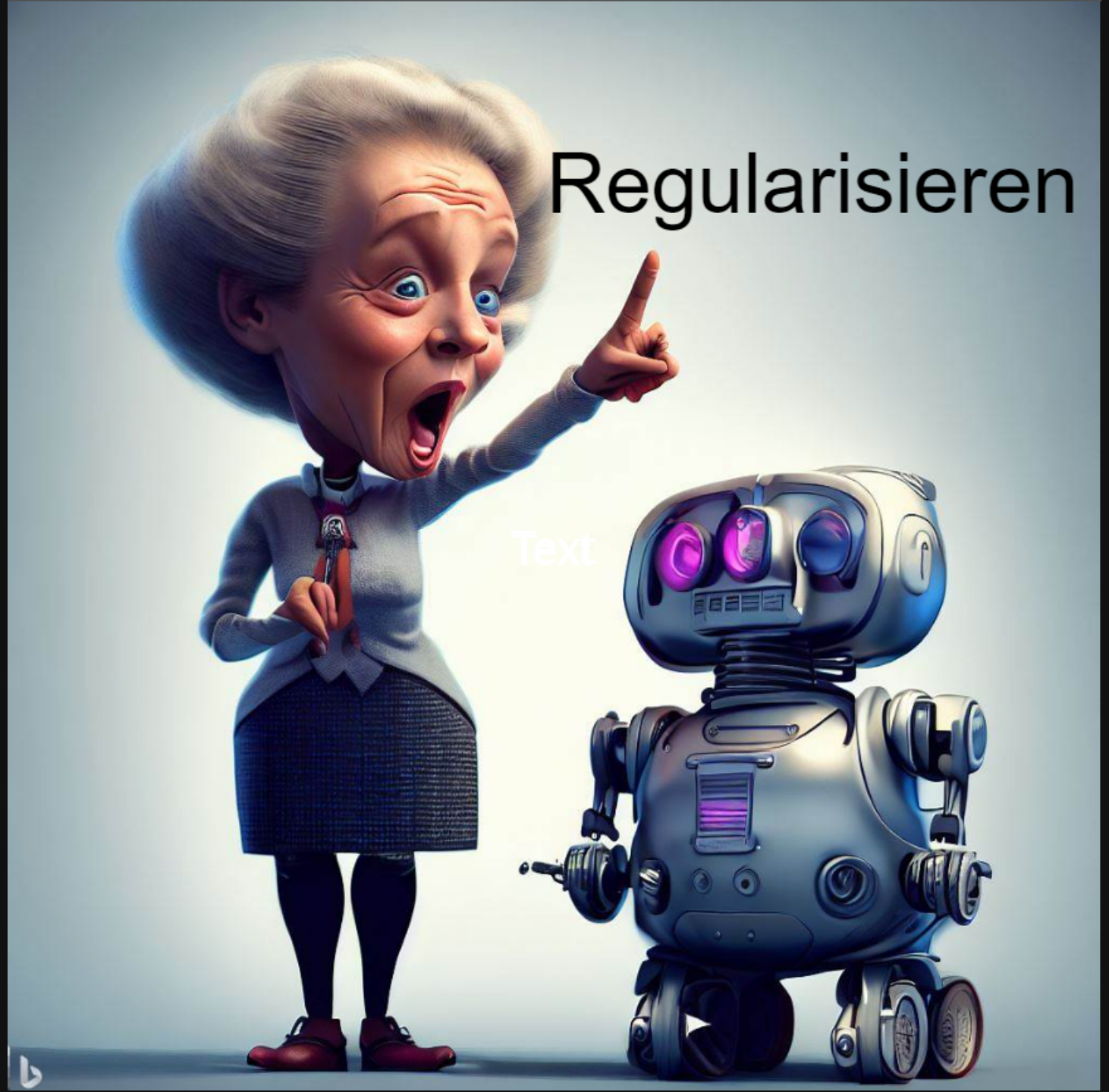
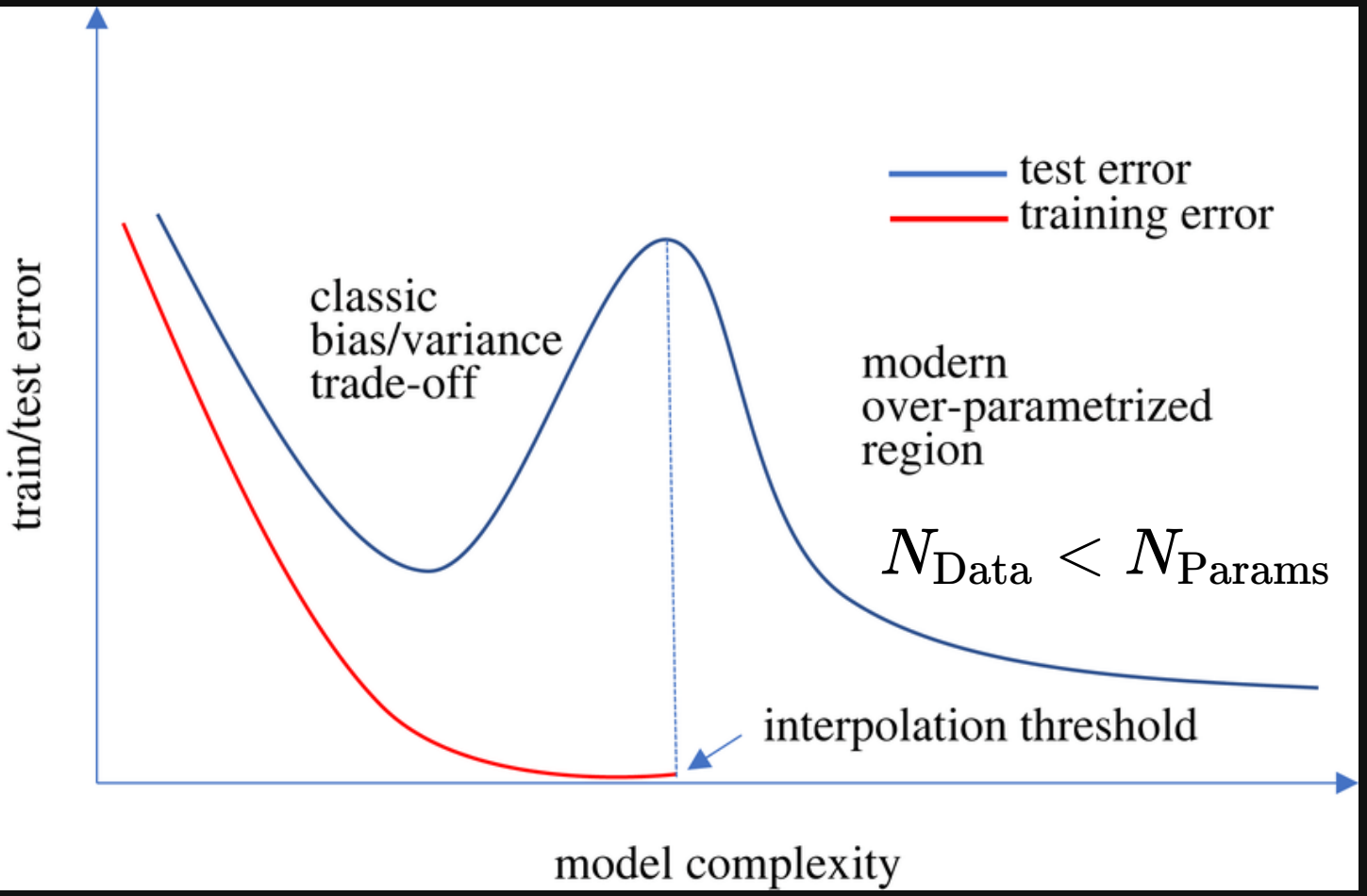

Regularisieren
Text

Wieso Regularisieren?



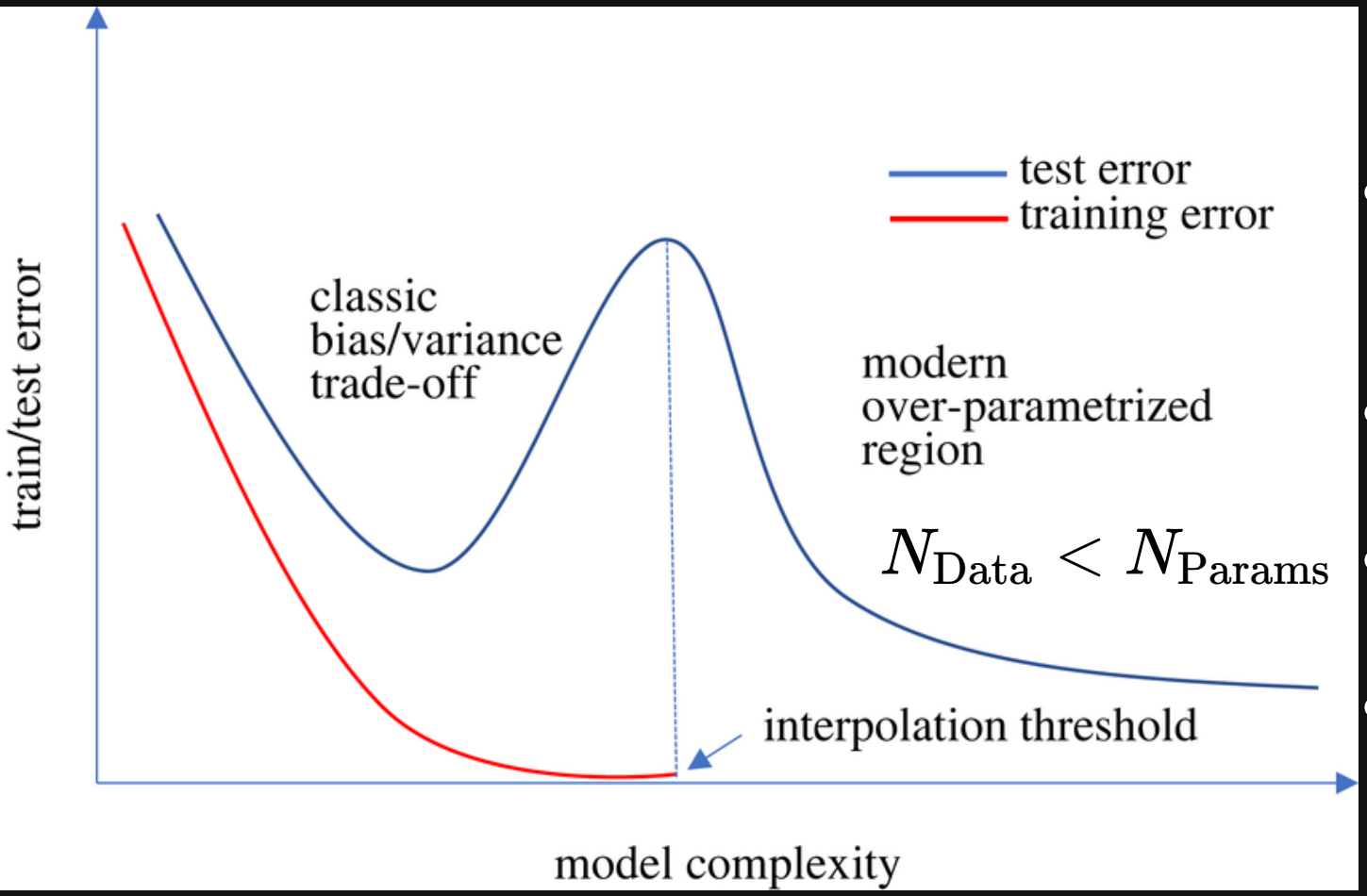
Tiefes NN kann jede Funktion fitten
-> overfit leicht
-> Generalisierung schwer

Regularisierung reduziert Varianz, ohne Bias zu erhöhen

-> Modell lernt "einfach" & robust sein

mehr zum Thema

Regularisieren



- reduziert Overfitting (essentiell bei grossen NN)
- verbessert Generalisierung
- robustere Models & Training
- Balance: Komplexität - Generalisierung

mehr zum Thema

Regularisieren

- L1 (Lasso): $\text{Loss} + \beta \sum_{i=1}^{n} |\theta_i|$
  - kleine Parameter
  - Gewichte -> 0 => Feature Auswahl

```
l1_regularization_loss = torch.norm(model.parameters(), 1)
```

Regularisieren

- L1 (Lasso): $\text{Loss} + \beta \sum_{i=1}^{n} |\theta_i|$
  - kleine Parameter
  - Gewichte -> 0 => Feature Auswahl

```
l1_regularization_loss = torch.norm(model.parameters(), 1)
```

- L2 (Ridge): $\text{Loss} + \beta \sum_{i=1}^{n} \theta_i^2$
  - kleine, ausgeglichene Parameter
  - meistgebrauchte regularisation

```
optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=beta)
```

Regularisieren

- L1 (Lasso): $\text{Loss} + \beta \sum_{i=1}^{n} |\theta_i|$
  - kleine Parameter
  - Gewichte -> 0 => Feature Auswahl

```
l1_regularization_loss = torch.norm(model.parameters(), 1)
```

- L2 (Ridge): $\text{Loss} + \beta \sum_{i=1}^{n} \theta_i^2$
  - kleine, ausgeglichene Parameter
  - meistgebrauchte regularisation

```
optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=beta)
```

- Dropout: setzt zufällig Neuronen auf 0
  - Modell redundant & robust

```
nn.Dropout(0.2)  # 0.2 of neurons set to 0
```

Regularisieren

- L1 (Lasso): $\text{Loss} + \beta \sum_{i=1}^{n} |\theta_i|$
  - kleine Parameter
  - Gewichte -> 0 => Feature Auswahl

```
l1_regularization_loss = torch.norm(model.parameters(), 1)
```

- L2 (Ridge): $\text{Loss} + \beta \sum_{i=1}^{n} \theta_i^2$
  - kleine, ausgeglichene Parameter
  - meistgebrauchte regularisation

```
optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=beta)
```

- Dropout: setzt zufällig Neuronen auf 0
  - Modell redundant & robust

```
nn.Dropout(0.2)  # 0.2 of neurons set to 0
```

- Batch Normalization: Normalisiert Layer Input
  - Reduziert Abhängikeit von vorigen Layern

```
nn.BatchNorm1d()
```

Regularisieren

- L1 (Lasso): $\text{Loss} + \beta \sum_{i=1}^{n} |\theta_i|$
  - kleine Parameter
  - Gewichte -> 0 => Feature Auswahl

```
Dense(N, kernel_regularizer=keras.regularizers.l1(beta), bias_regularizer=keras.regularizers.l1(beta))
```

- L2 (Ridge): $\text{Loss} + \beta \sum_{i=1}^{n} \theta_i^2$
  - kleine, ausgeglichene Parameter
  - meistgebrauchte regularisation

```
Dense(N, kernel_regularizer=keras.regularizers.l2(beta), bias_regularizer=keras.regularizers.l2(beta))
```

- Dropout: setzt zufällig Neuronen auf 0
  - Modell redundant & robust

```
tf.keras.layers.Dropout(0.2)  # 0.2 of neurons set to 0
```

- Batch Normalization: Normalisiert Layer Input
  - Reduziert Abhängikeit von vorigen Layern

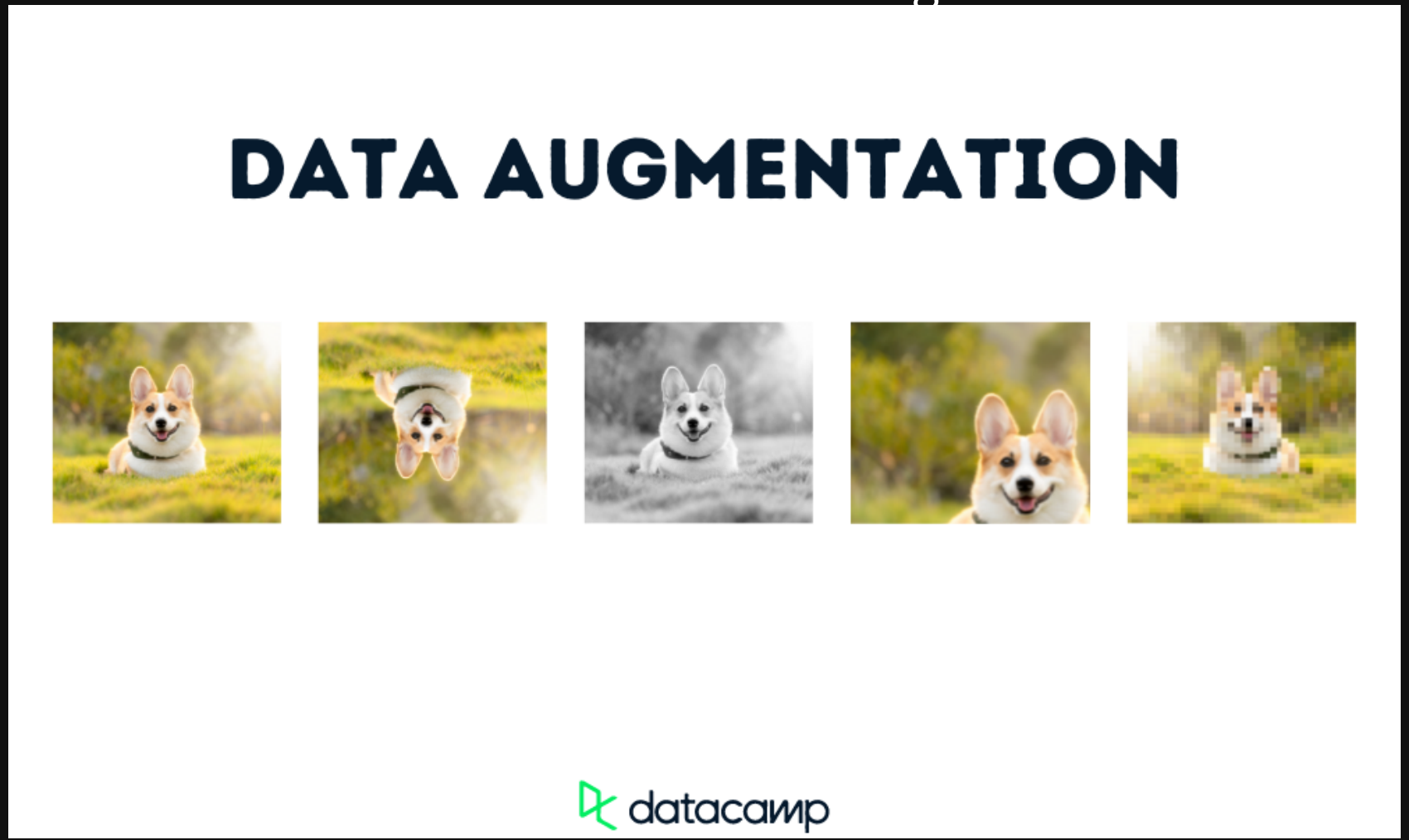
```
tf.keras.layers.BatchNormalization(),
```

Data Augmentation

- Mehr Daten -> grössere Modelle -> komplexere Aufgaben

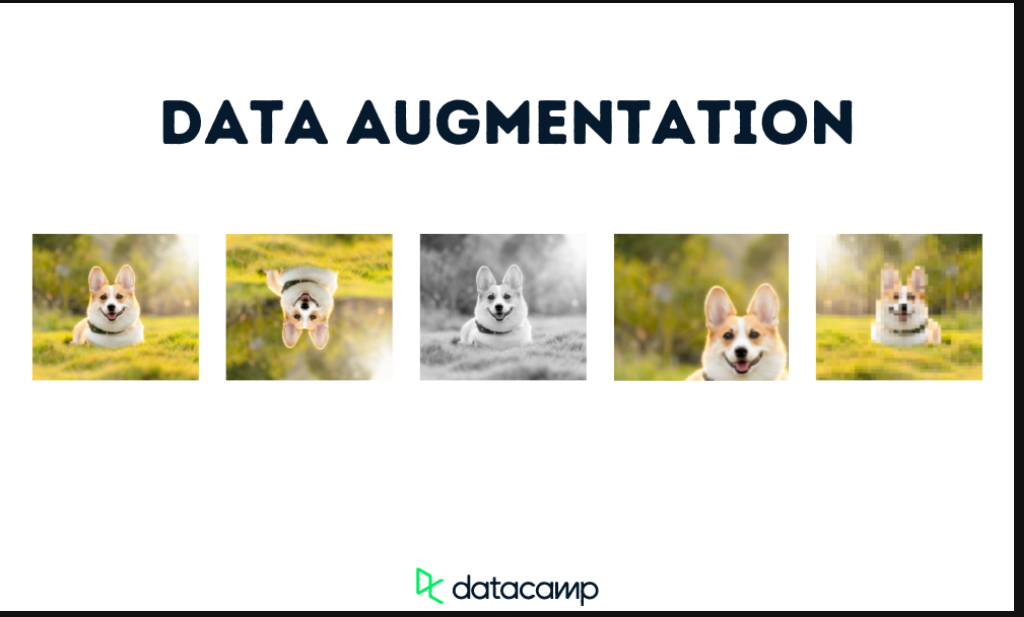
Data Augmentation

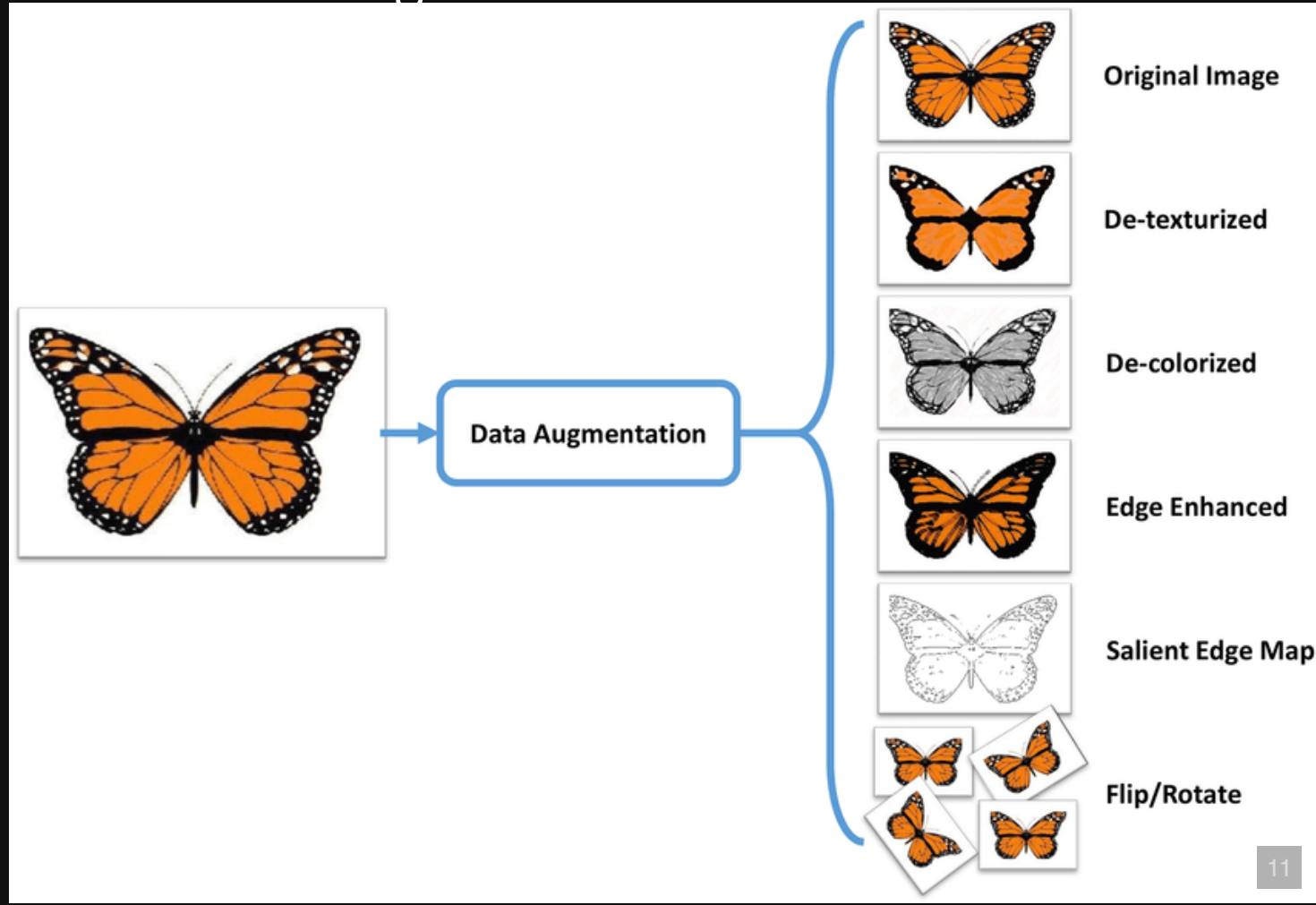
- Mehr Daten -> grössere Modelle -> komplexere Aufgaben
- Mehr Daten durch Datenbearbeitung



Data Augmentation

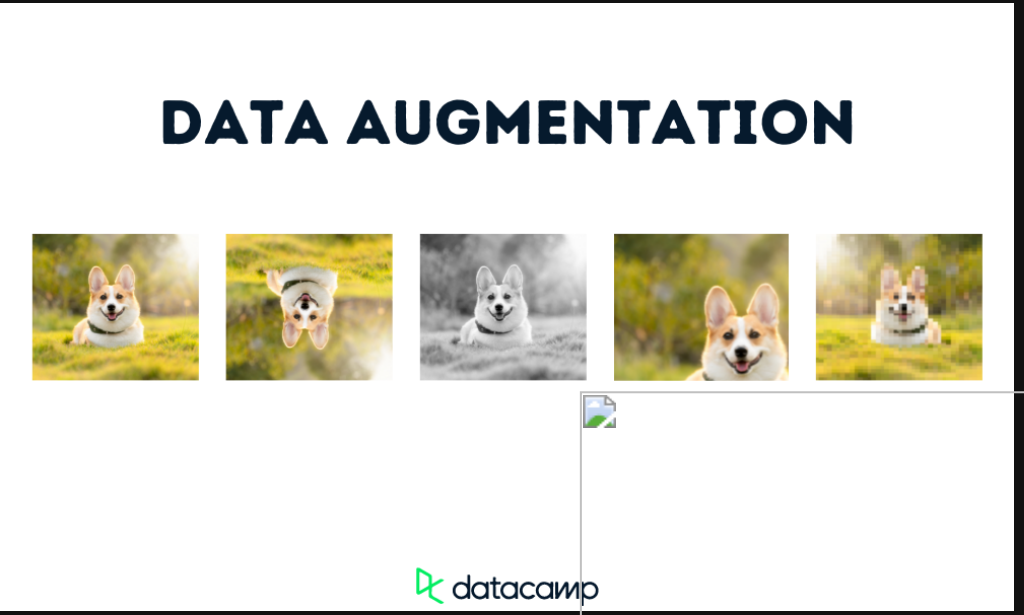
- Mehr Daten -> grössere Modelle -> komplexere Aufgaben
- Mehr Daten durch Datenbearbeitung

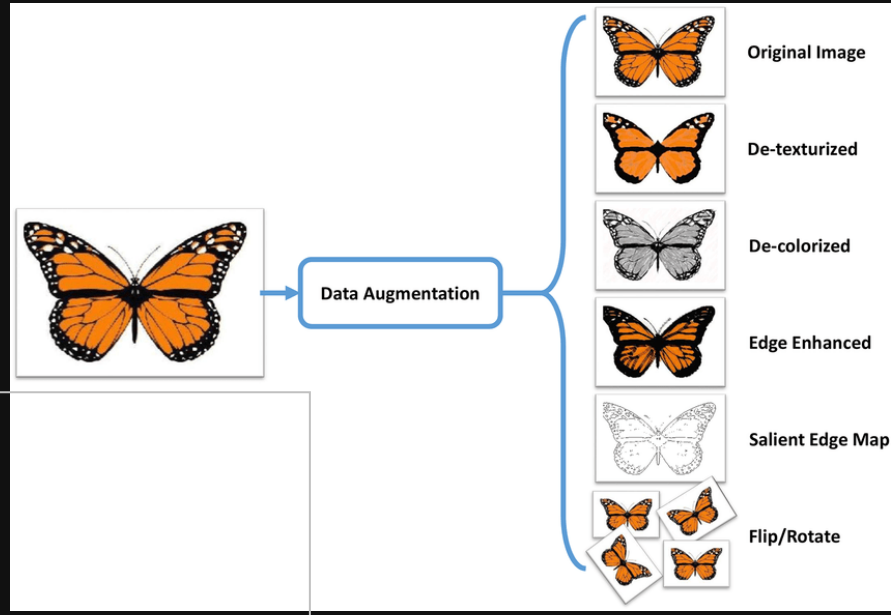




Data Augmentation

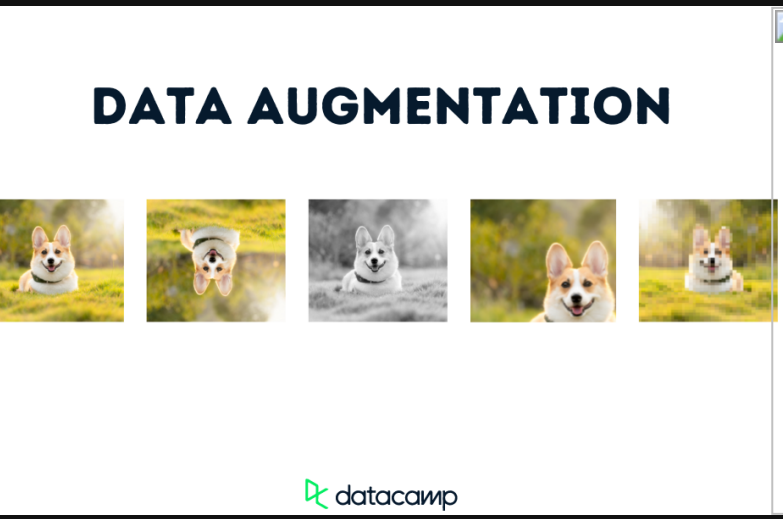
- Mehr Daten -> grössere Modelle -> komplexere Aufgaben
- Mehr Daten durch Datenbearbeitung

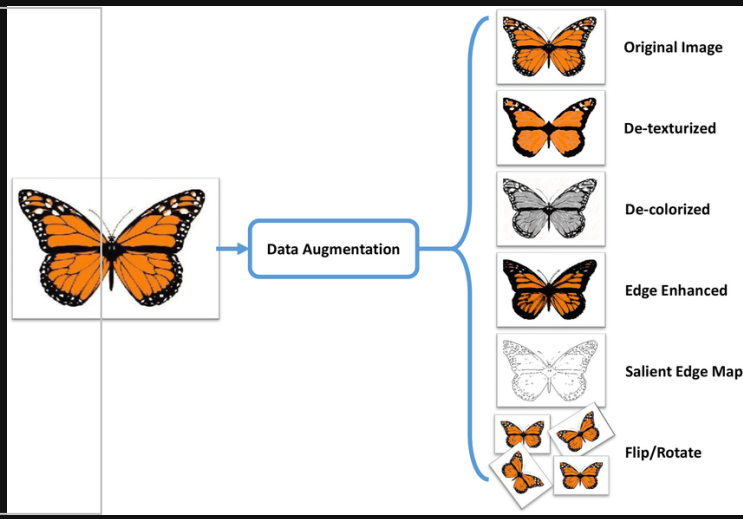




Data Augmentation

- Mehr Daten -> grössere Modelle -> komplexere Aufgaben
- Mehr Daten durch Datenbearbeitung
- Verbessert Generalisierung
- Reduziert Overfitting





Mehr zu Image Augmentation

Regularisieren

Hands-On: MNIST Classifier

Bearbeiten Sie dieses Notebook

- Modifizieren Sie den MNIST Classifier mit zwei Regularisierungen: Dropout & L2
- Vergleichen Sie die Performance mit und ohne Regularisierung

Die Lösung finden Sie in diesem Notebook