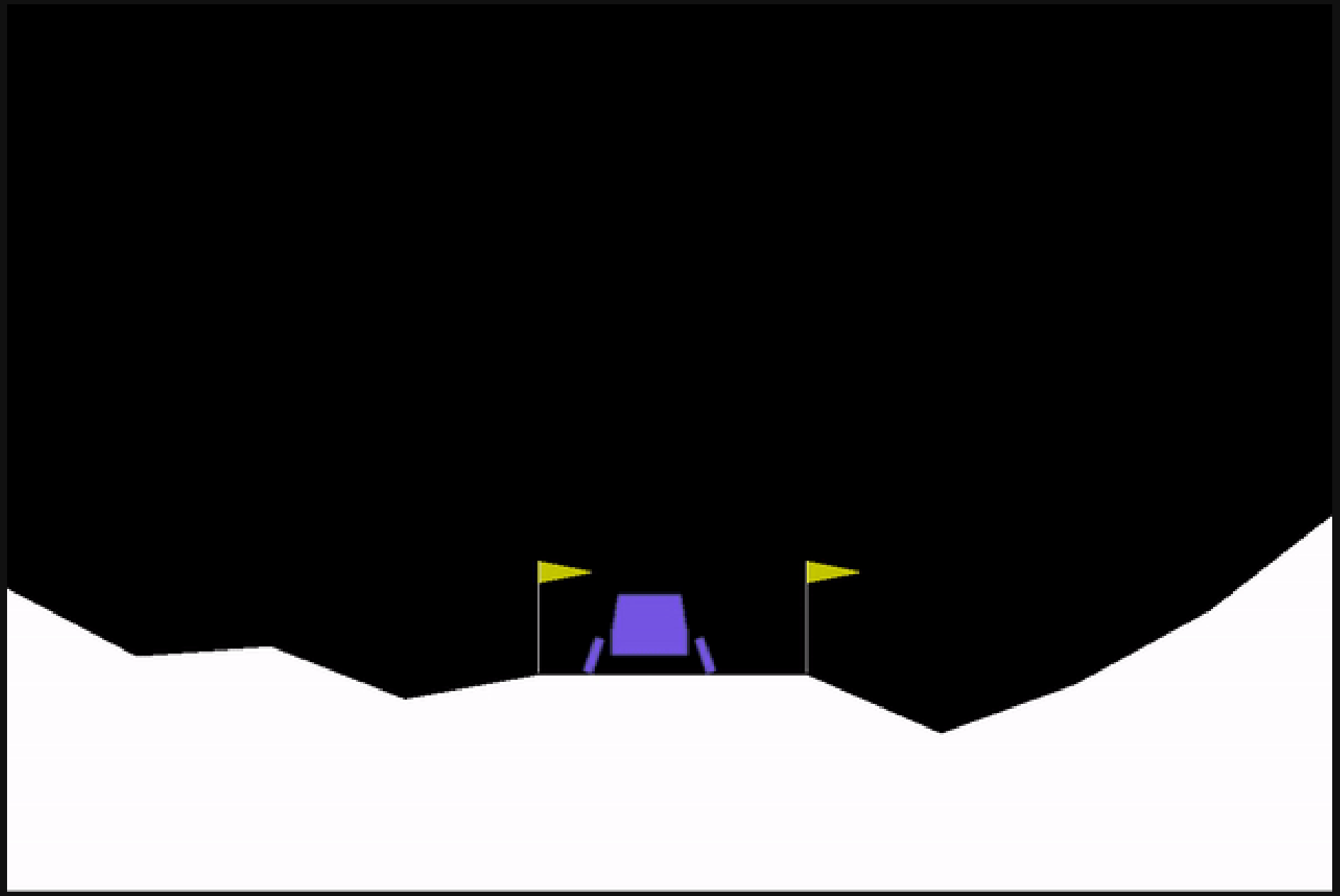

Optimieren

Optimieren



Optimieren

Modell auf unbekannten Daten optimieren

aber

Testset erst ganz zum Schluss verwenden

Modell auf unbekannten Daten optimieren

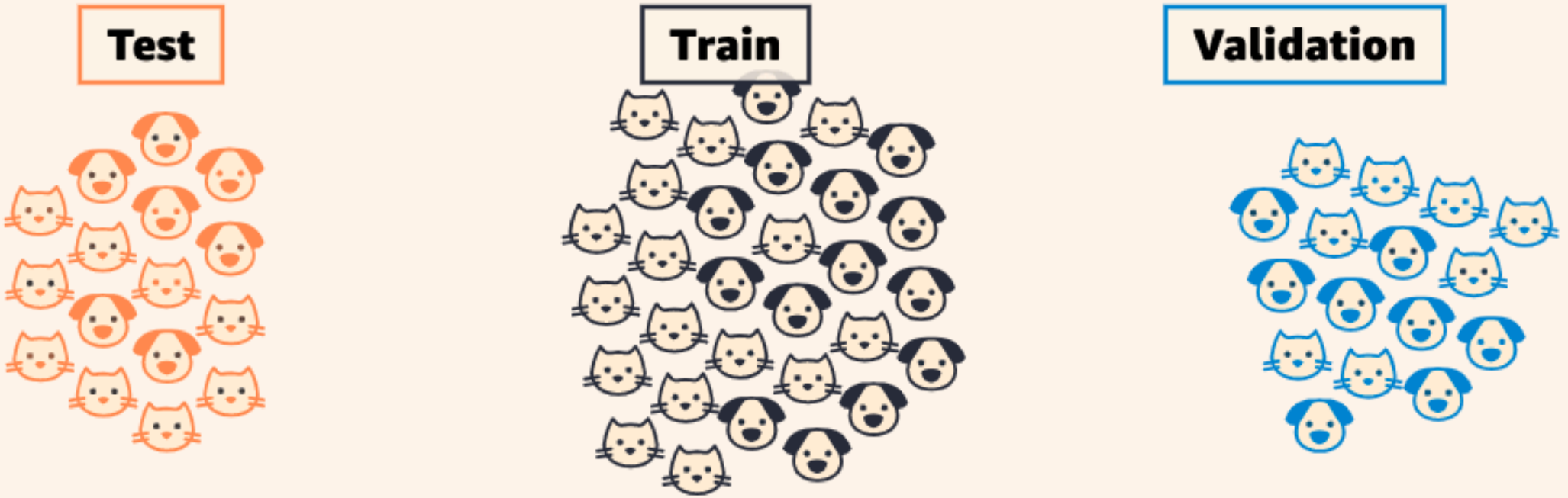
aber

Testset erst ganz zum Schluss verwenden

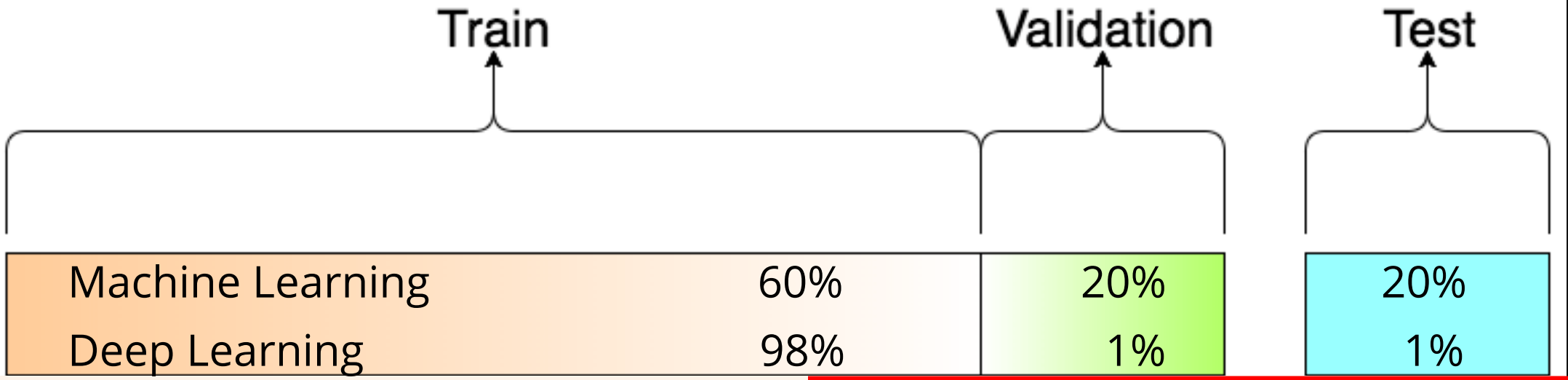
Weiteres Datenset absplitten

Train-Test-Valid Split

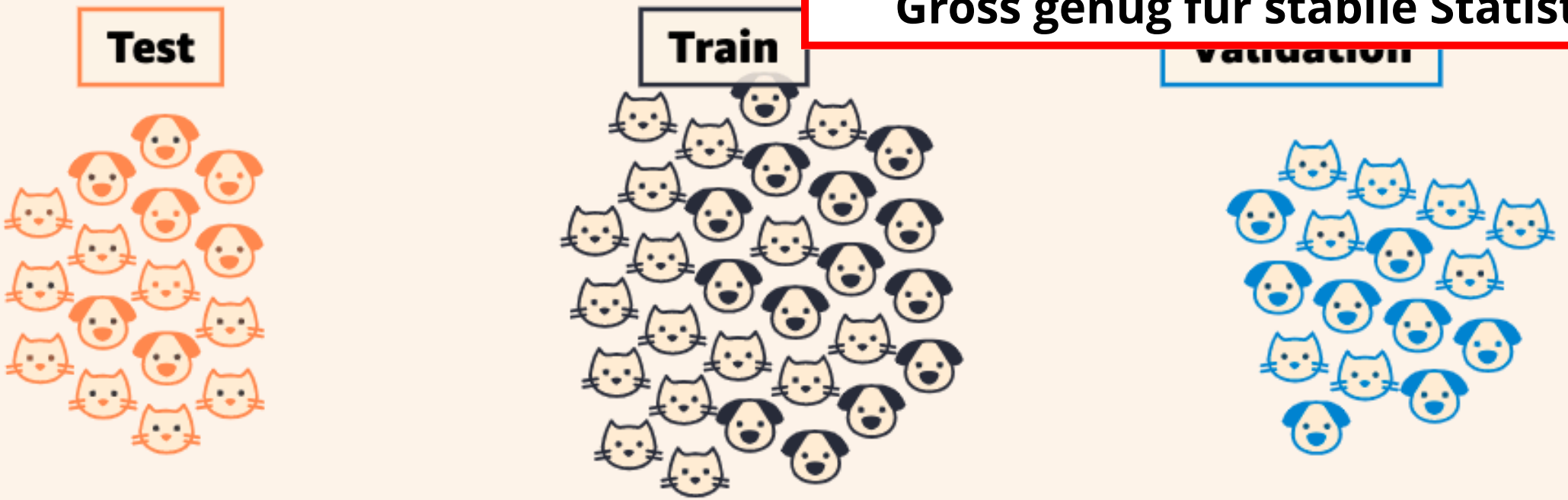
Train-Test-Valid Split



Train-Test-Valid Split



Gross genug für stabile Statistik



Train-Test-Valid Split

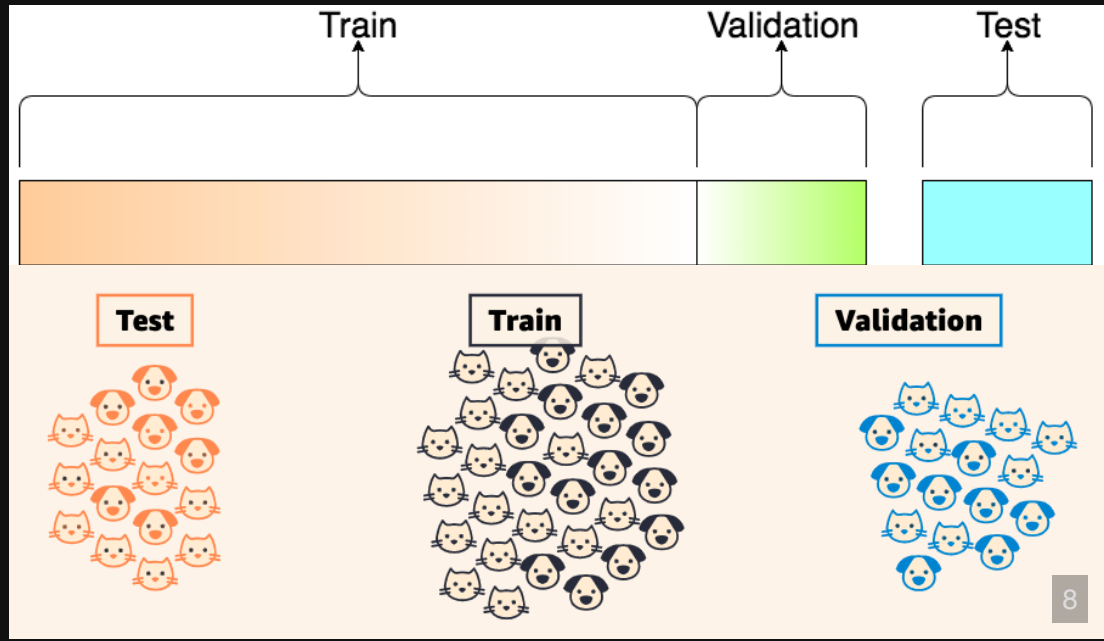
1. verschiedene Settings wählen (Hyperparameter / Architektur)

2. für wenige Epochen trainieren

3. Leistung auf Validationsdaten vergleichen

4. Setting mit bester Leistung voll trainieren

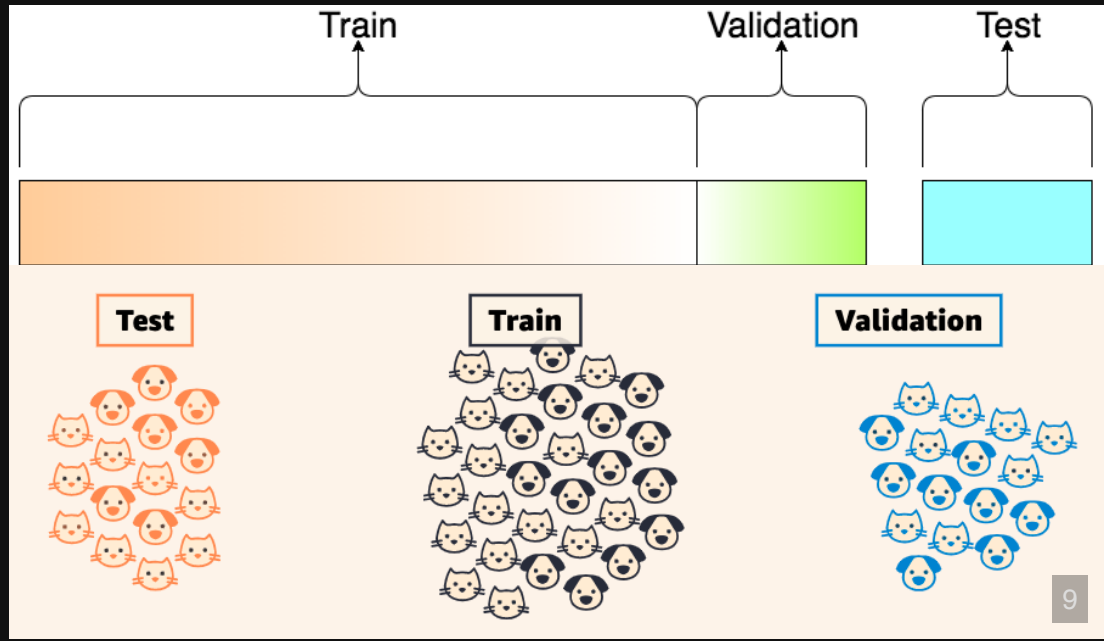
5. Trainiertes Modell auf Testset Evaluieren



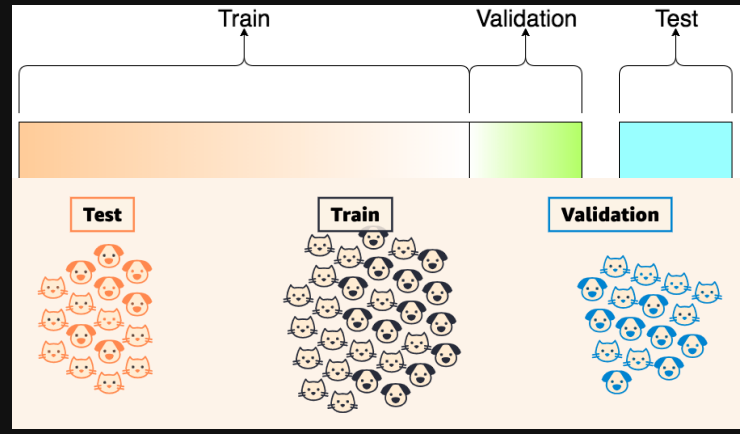
Train-Test-Valid Split

1. verschiedene Settings wählen (Hyperparameter / Architektur)
2. für wenige Epochen trainieren
3. Leistung auf Validationsdaten vergleichen
4. Setting mit bester Leistung voll trainieren
5. Trainiertes Modell auf Testset Evaluieren

```python
from torch.utils.data import random_split
train_ratio, valid_ratio = 0.8, 0.2

# Gesamteanzahl der Trainingsdaten
N_training = len(training_data)

# Berechne die Anzahl der Beispiele für jeden Split
train_size = int(train_ratio * N_training)
valid_size = N_training - train_size

# Teile den Trainingsdatensatz in Train und Valid auf
train_data, valid_data = random_split(training_data,
                                      [train_size, valid_size])
```



Train-Test-Valid Split



1. verschiedene Settings wählen (Hyperparameter / Architektur)
2. für wenige Epochen trainieren
3. Leistung auf Validationsdaten vergleichen
4. Setting mit bester Leistung voll trainieren
5. Trainiertes Modell auf Testset Evaluieren

```
orch.utils.data import random_split
ratio, valid_ratio = 0.8, 0.2

nteanzahl der Trainingsdaten
ning = len(training_data)

chne die Anzahl der Beispiele für jeden Split
size = int(train_ratio * N_training)
size = N_training - train_size

e den Trainingsdatensatz in Train und Valid auf
data, valid_data = random_split(training_data,
                                [train_size, valid_size])
```

```
1  import tensorflow as tf
2  from sklearn.model_selection import train_test_split
3  valid_ratio = 0.2
4
5  input_data = your_input_data
6  labels = your_labels
7
8  input_train, input_valid, labels_train, labels_valid =
9          train_test_split(input_data, labels, test_size=valid_ratio, random_st
10
11 # Create TensorFlow Datasets
12 train_dataset = tf.data.Dataset.from_tensor_slices((input_train, labels_train
13 valid_dataset = tf.data.Dataset.from_tensor_slices((input_valid, labels_valid
```

Hyperparameter

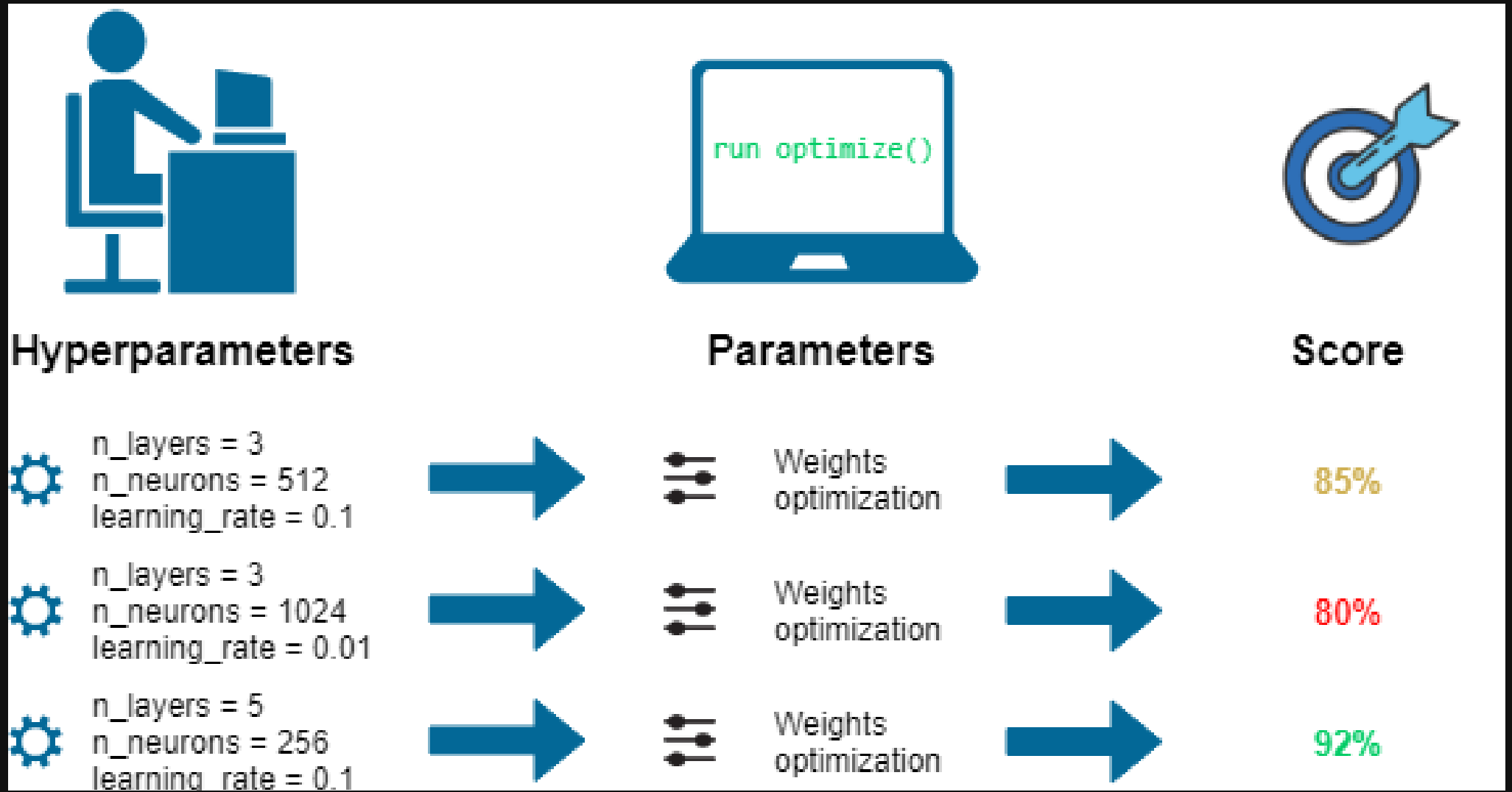
Hyperparameter

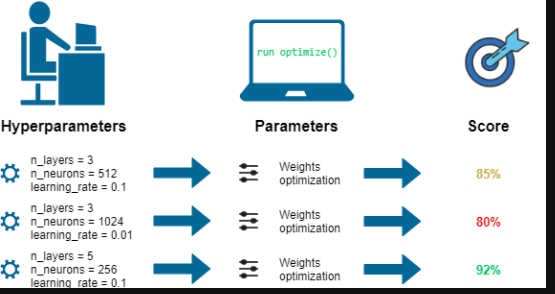
```python
class MLP_var(nn.Module):
    def __init__(self, N_layer: int):
        super(MLP_var, self).__init__()
        layers = []
        layers.append(nn.Linear(28*28, 64))
        layers.append(nn.ReLU())
        for _ in range(N_layer-1):
            layers.append(nn.Linear(64, 64))
            layers.append(nn.ReLU())
        layers.append(nn.Linear(64, 10))
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.model(x)
        return x
```

Hyperparameter Search

Hyperparameter Search

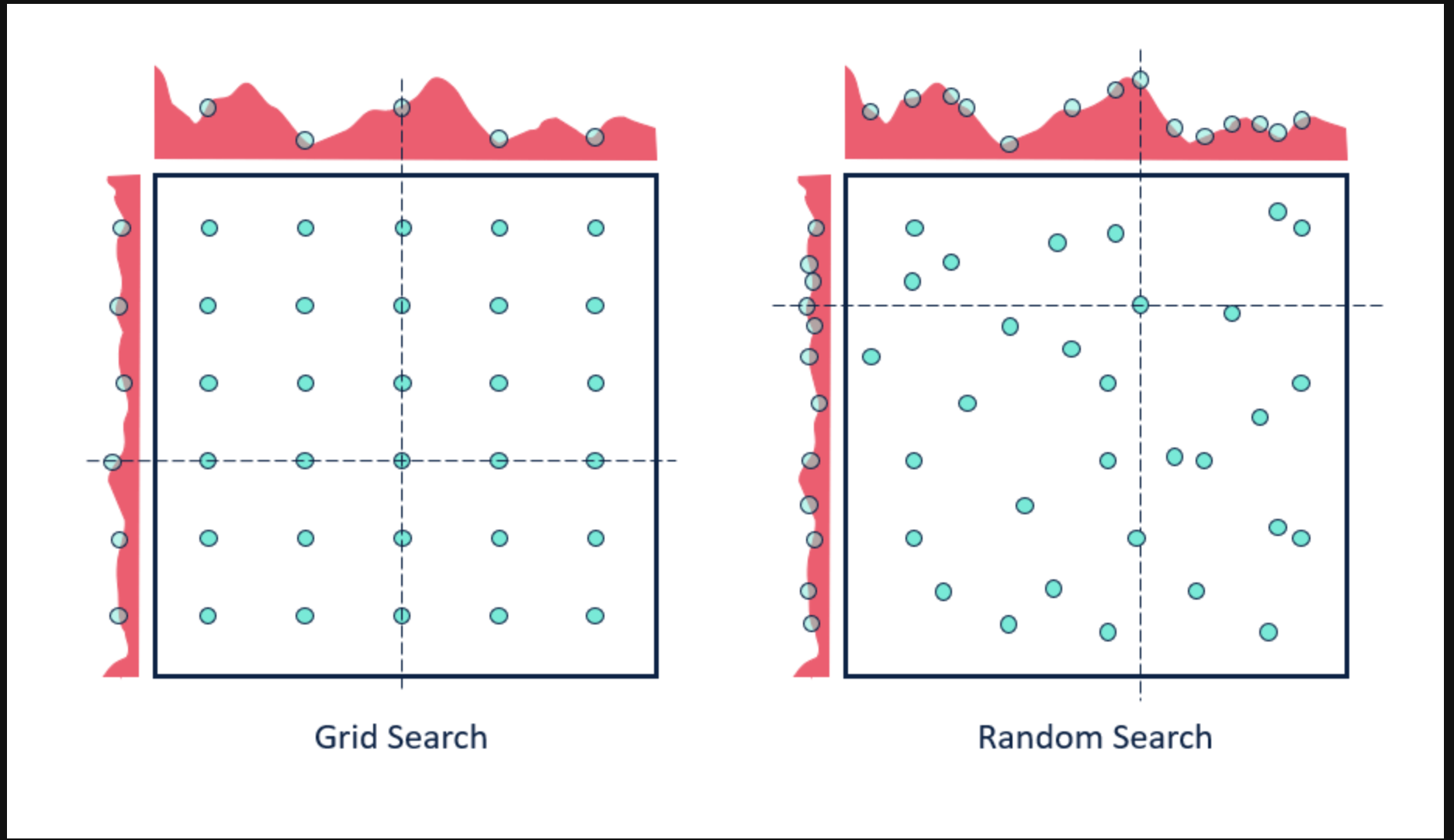
Try different Values, pick best score

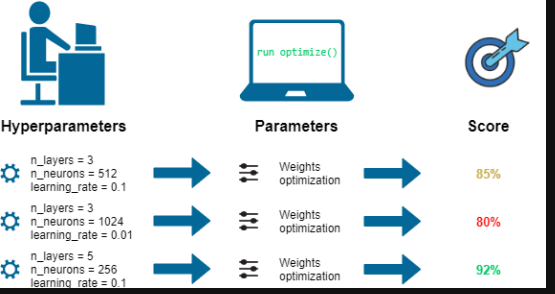




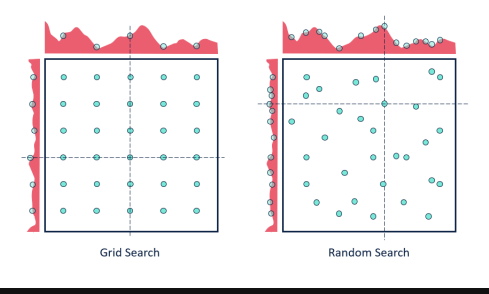
Hyperparameter Search

Try different Values, pick best score





Hyperparameter Search

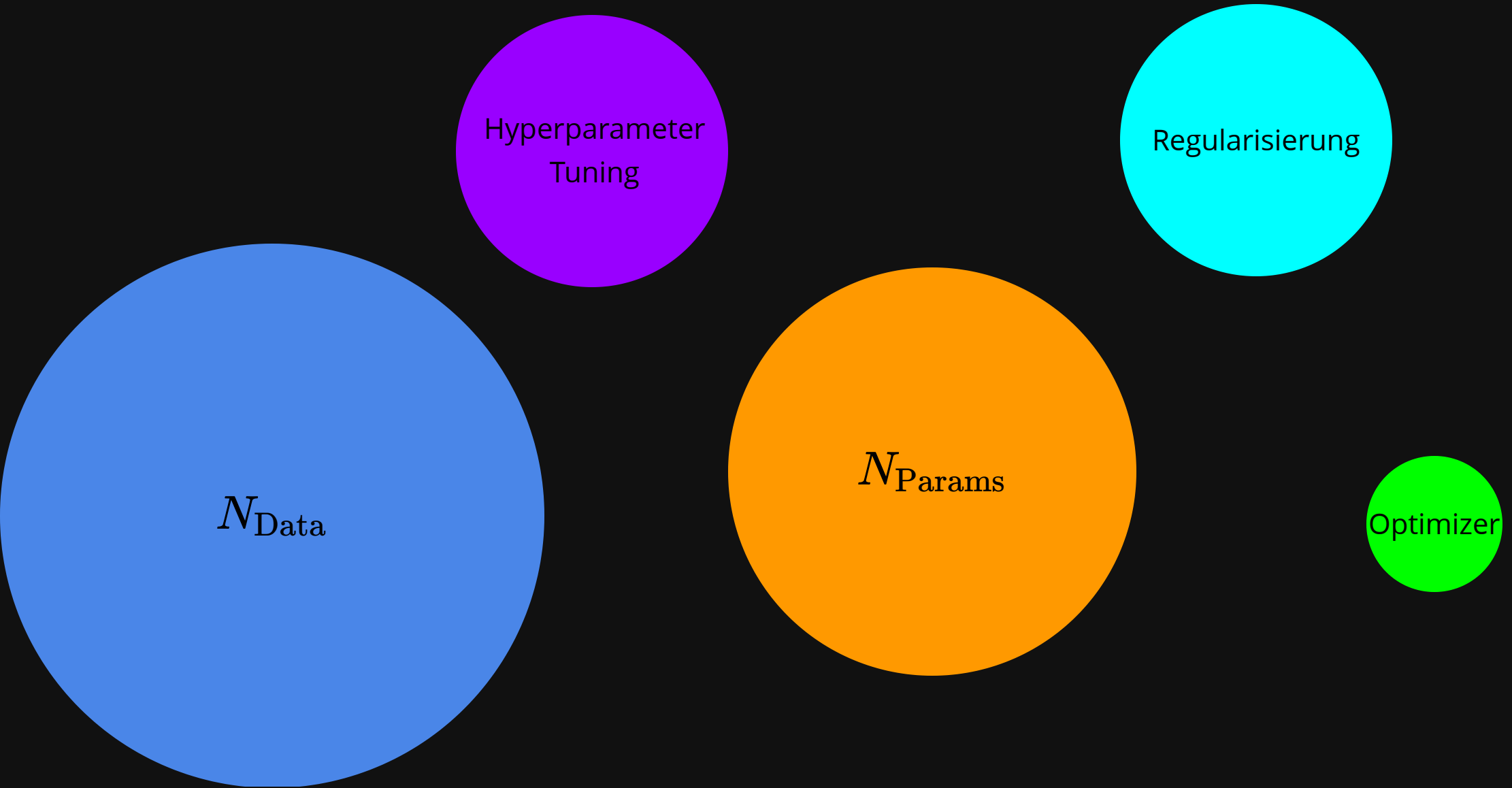


Gridsearch

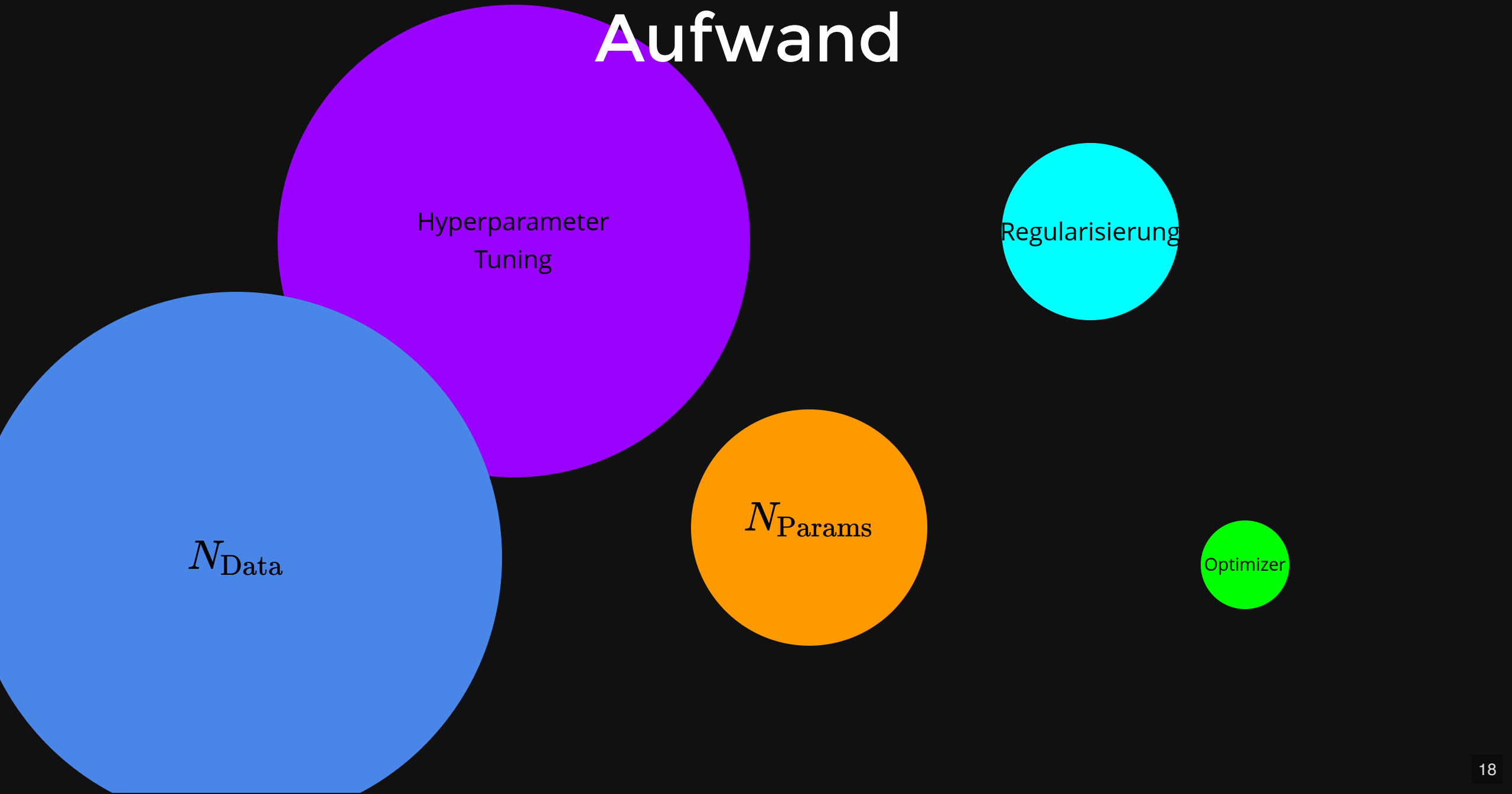
```python
for (N_layer, lr) in product(N_layer_values, learning_rate_values):
    model = MLP_var(N_layer)
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    losses_train, losses_valid, f1_scores_train, f1_scores_valid = full_training(
      model, criterion, optimizer, training_loader, valid_loader, epochs=epochs
    )
```

weitere Tuning Methoden

Einfluss auf Stabilität



Aufwand


Hyperparameter Tuning
Regularisierung
N_{Data}
N_{Params}
Optimizer

Optimieren

Hands-On: MNIST Classifier

Bearbeiten Sie dieses Notebook

- Erstellen Sie einen MNIST Classifier mit variabler Anzahl hidden Layer
- Erstellen Sie einen Validationsdatensatz
- Führen sie ein Hyperparametertuning durch

Die Lösung finden Sie in diesem Notebook