Friedrich-Alexander-Universität
Erlangen-Nürnberg

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

Automatic hyperparameter optimization for Graph Neural Networks

Lukas Hennig
May 03, 2023
GraphNeT III

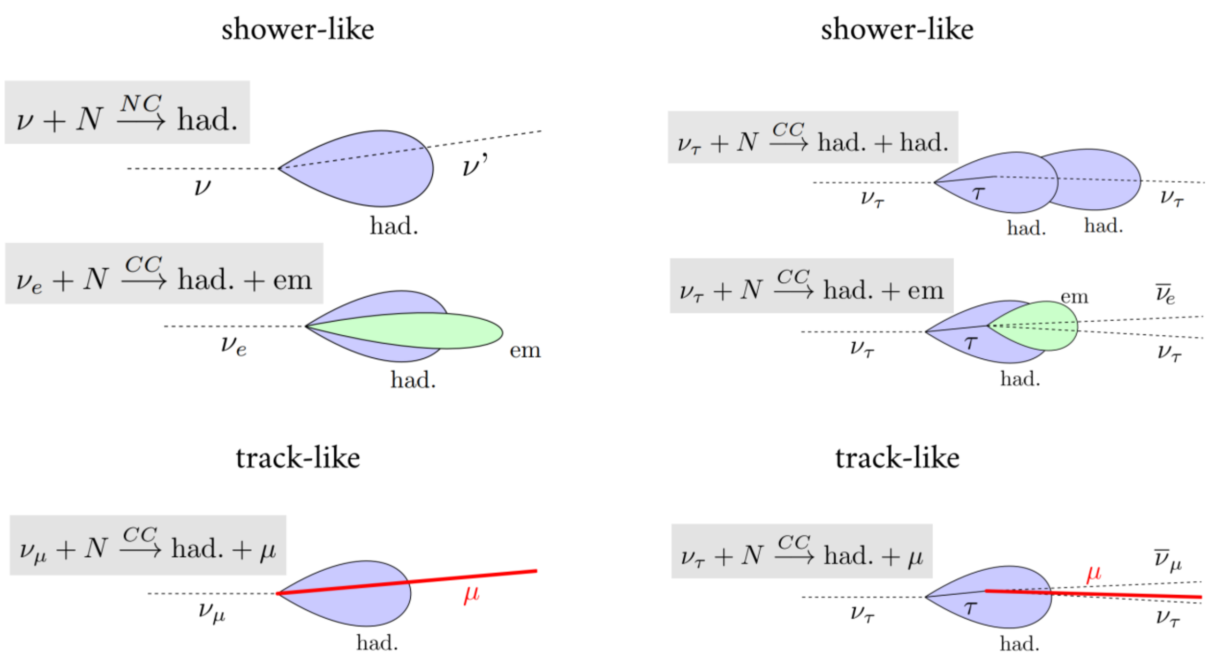
Motivation

- Goal of my master thesis: Identifying tau neutrino events in the KM3NeT/ORCA detector with GNNs
- ORCA is sensitive to atmospheric neutrinos
- Tau neutrinos not produced in atmosphere, detected tau neutrinos exist due to neutrino oscillation
- GNNs are trained and evaluated on Monte-Carlo simulations
- Evaluation uses MC events weighted according to Honda neutrino flux

$\Rightarrow$ Tau neutrinos are the minority class

- Same for training, but tau neutrino events scaled up to achieve class balance

Nontau events **Tau events**



Status of my master thesis

- Applied different optimization steps:
  - Generated more simulation data ✓
  - Tried out different weightings ✓
  - Manual hyperparameter optimization ✓
- Currently ongoing, final step consists of an **automatic hyperparameter optimization (AHPO)**
- Two main goals with AHPO
  - Optimizing the performance of the GNNs by searching for optimal hyperparameter configuration
  - By trying out different hyperparameter configurations, one could learn something about which hyperparameters are the most relevant, if there are some correlations between different hyperparameters, etc.
- AHPO uses algorithms designed for “finding good hyperparameters with as few trainings as possible”

Ray Tune

- I am using Ray Tune as AHPO framework
- Ray is a framework for scaling Machine Learning tasks up for use on distributed systems
- Ray Tune is the part of the framework dealing with AHPO
- Many pre-implemented AHPO algorithms
- **Search algorithms**: suggests next hyperparameter configuration to evaluated
  - Grid search, Random search, Bayes Optimization search, …
- **Scheduler algorithms:** determines which configurations get computing resources, early termination of bad configs
  - Hyperband, Population based training, Asynchronous successive halving algorithm (ASHA), …

https://www.ray.io/

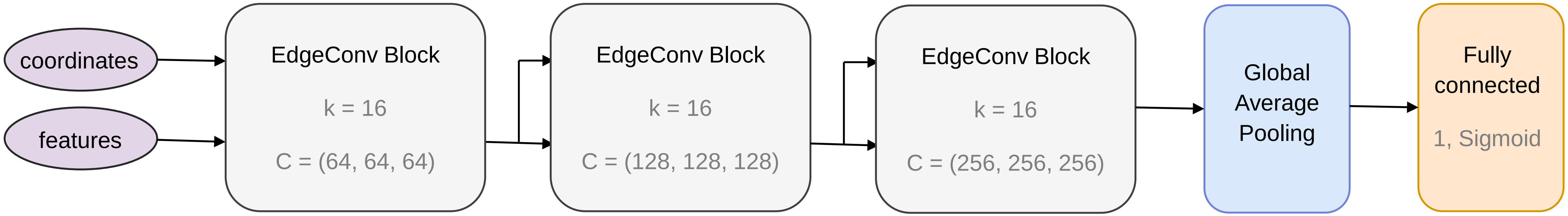
Ray Tune

- I am using **Random Search** for suggestion of hyperparameter configs and **ASHA** for scheduling
- This is the "go-to solution" recommended in Ray Tunes FAQ for smaller problems
- ASHA assigns each trial, i.e., each suggested hyperparameter configuration to a so called **"bracket"**
- Each bracket has some epoch checkpoints where the performance of the trained models is compared
- Only the top 50% of networks is allowed to continue training after a checkpoint is reached, the other half is terminated, and a new trial is started

```
Bracket: Iter 32.000: None | Iter 16.000: None | Iter 8.000: None | Iter 4.000: None
Bracket: Iter 32.000: None | Iter 16.000: None | Iter 8.000: None
Bracket: Iter 32.000: None | Iter 16.000: None
```

Properties of the GNNs

- Our GNN architecture is based on **ParticleNet** and implemented in OrcaNet
- The **EdgeConv Block** maps a graph with F features per node to a graph with the same number of nodes, but with F' features per node
- Each node has information about a Cherenkov light signal associated with a triggered event as its features, e.g., the position and time
- A subset of these features is used as coordinates to calculate a nodes' k nearest neighbors with a predefined distance measure
- Message passing is implemented on a node level using a shared multilayer perceptron (details on the next slide)



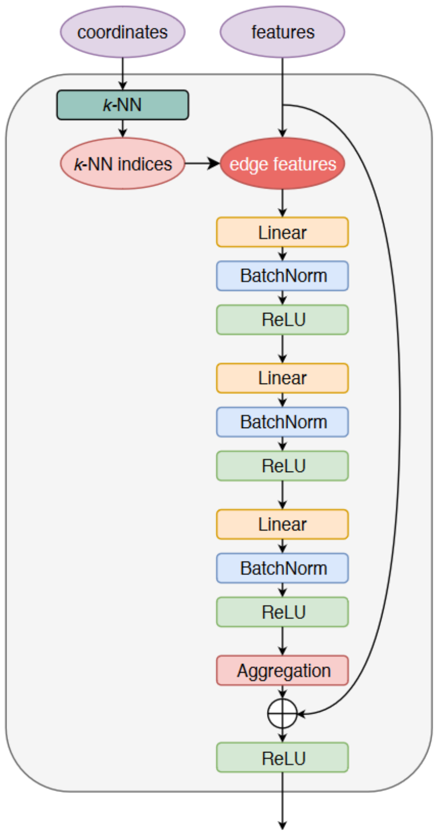
EdgeConv block

Mathematical description:

- Given: node i with its k nearest neighbors i_j with $j \in \{0, \ldots, k-1\}$
- Features of node i are denoted $x_i \in \mathbb{R}^{\mathrm{F}}$
- Given an edge function $h_\Theta\left(x_i, x_{i_j} - x_i\right) : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^{F'}$ with trainable weights Θ, implemented as MLP
- Perform a convolution over the nearest neighbors:

$$x_i' = \frac{1}{k} \sum_{j=0}^{k-1} h_\Theta\left(x_i, x_{i_j} - x_i\right)$$

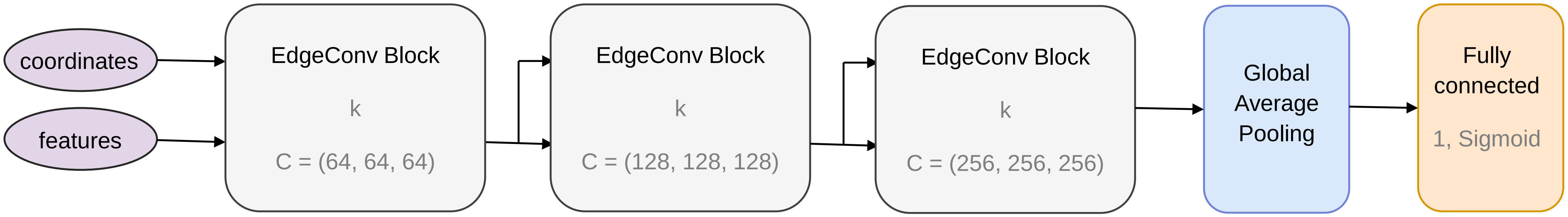
Implementation:



Manual HPO

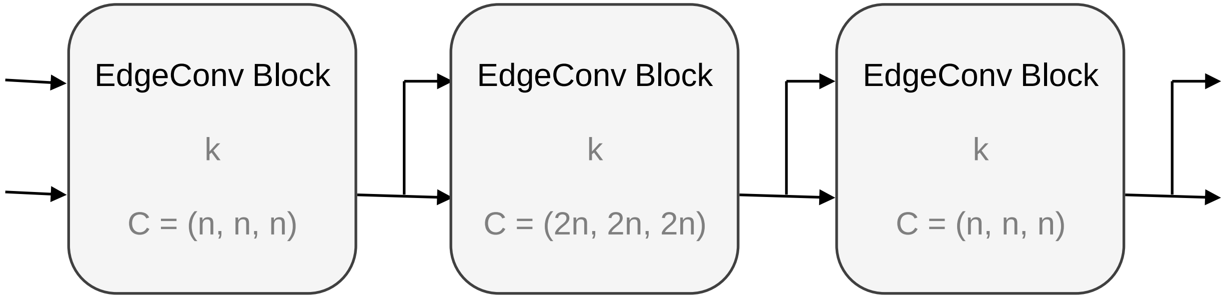
- Architecture below was used for manual HPO where the number of neurons in MLP is doubled after each block
- Between one and three dense layers before output layer, decreasing by a factor of two per layer
- Varying a few parameters from architecture below, e.g. number of EdgeConv blocks and dense layers, number of nearest neighbors k, learning rate, and batchsize
- Chosen evaluation metric (PR-AUC) had value up to 0.174 (for later comparison)

edgeconvLayers	denseLayers	best_epoch	pr_auc
5	2	14	0.17496501
3	2	29	0.17426357
5	1	13	0.17393218
4	1	24	0.17387258
4	1	22	0.17284074
5	3	14	0.17237942
4	2	21	0.17233823
4	2	27	0.17204231
5	2	15	0.1713409



First AHPO

- Two build methods used for first AHPO
- First method called “per_block”, indicating that exponential increase in MLP neurons happens after a block
- After reaching a “peak width” in an EdgeConv block, further EdgeConv blocks are allowed with decreasing number of neurons (in contrast to manual HPO)
- Parameter space shown on the right

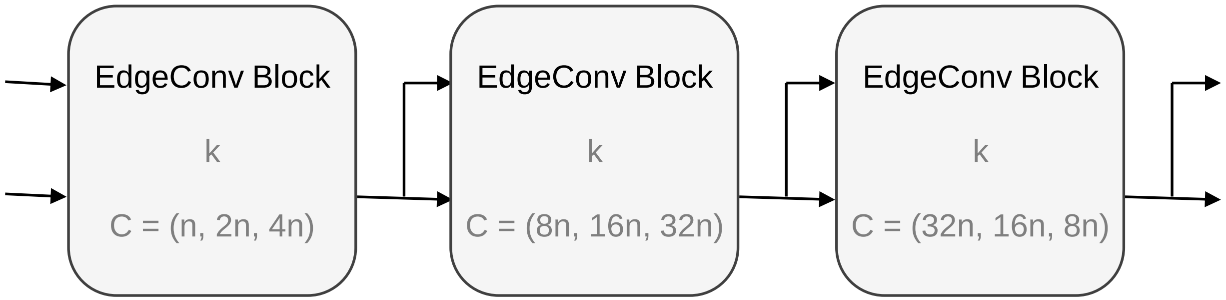



```python
class ParamSpace:
    def __init__(self) -> None:
        self.learning_rate = tune.qloguniform(1e-5, 1e-1, 5e-6)
        self.use_lr_schedule = tune.choice([True, False])
        self.batchsize = tune.choice([16, 32])
        self.shortcut = tune.choice([True, False])
        self.batchnorm = tune.choice([True, False])
        self.useDropout = tune.choice([True, False])
        self.dropout_rate = tune.sample_from(get_dropout_rate)
        self.kNN = tune.choice([25, 30, 35, 40, 45])


class ParamSpacePerBlock(ParamSpace):
    def __init__(self) -> None:
        super().__init__()
        self.number_kernel_network_nodes_begin = tune.choice([24, 26, 28, 30])
        self.number_kernel_units = tune.choice([2, 3, 4])
        self.number_edgeconvs_until_peak = tune.choice([5, 6])
        self.number_edgeconvs_after_peak = tune.sample_from(
            lambda spec: np.random.randint(0, spec.config.number_edgeconvs_until_peak)
        )
        self.number_dense_layers = tune.sample_from(
            lambda spec: np.random.randint(
                0,
                spec.config.number_edgeconvs_until_peak
                - spec.config.number_edgeconvs_after_peak,
            )
        )

```

First AHPO

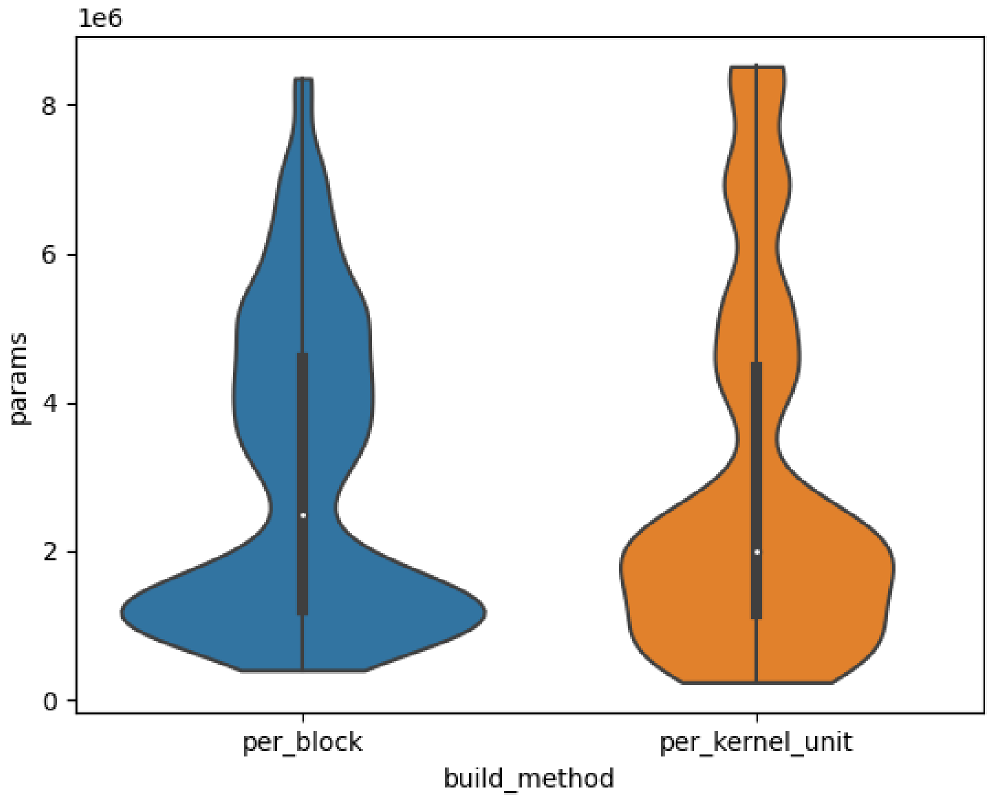
- Second build method is called “per_kernel_unit”, indicating that an exponential increase happens in each layer of the MLP
- Exponential increase by a factor of two in beginning, later changed to sqrt(2)
- EdgeConv blocks after the peak width are decreasing in the same way

```
class ParamSpacePerKernelUnit(ParamSpace):
    def __init__(self) -> None:
        super().__init__()
        self.number_kernel_network_nodes_begin = tune.choice([16, 24, 30])
        self.number_kernel_units = tune.choice([2, 3])
        self.number_edgeconvs_until_peak = tune.sample_from(
            get_number_edgeconvs_until_peak_for_per_kernel_unit
        )
        self.number_edgeconvs_after_peak = tune.sample_from(
            lambda spec: np.random.randint(0, 4)
        )
        self.number_dense_layers = tune.sample_from(
            lambda spec: np.random.randint(
                0,
                4 - spec.config.number_edgeconvs_after_peak,
            )
        )
```



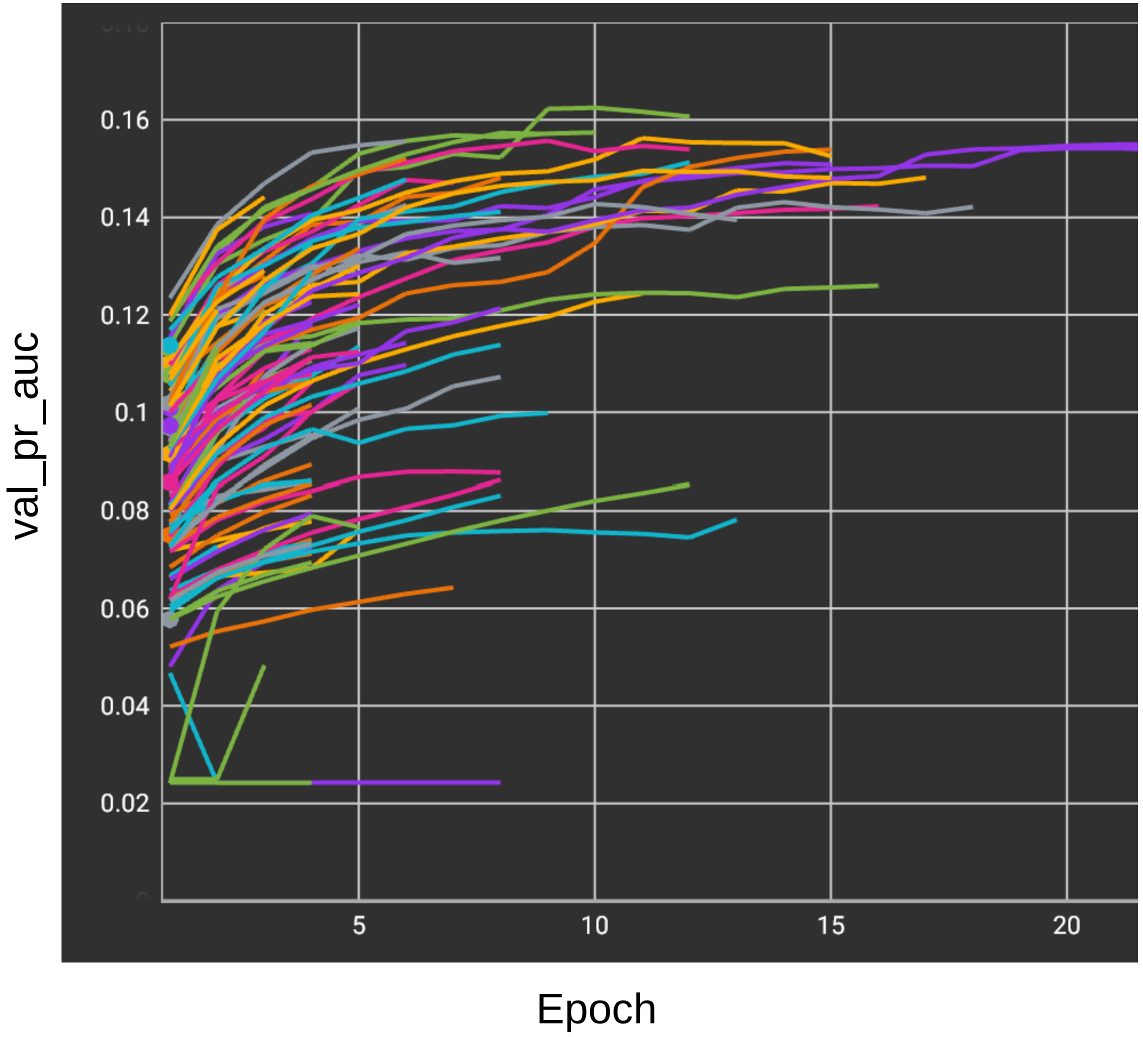
First AHPO

- Ranges for each hyperparameter and for each build method were chosen manually to obtain networks with number of parameters up to about 9 million
- Otherwise, time for training would be too long



Results of first AHPO

- On the right, the PR-AUC on the validation data for the trained GNNs can be seen
- 198 GNNs were trained during first AHPO
- GNNs performed worse than during manual HPO, where all models reached PR-AUC of over 0.16, many after about 15-20 epochs
- Do the exponentially decreasing EdgeConv blocks introduce a bottleneck? Is the MLP for the first EdgeConv blocks too small, which could cause a bottleneck?



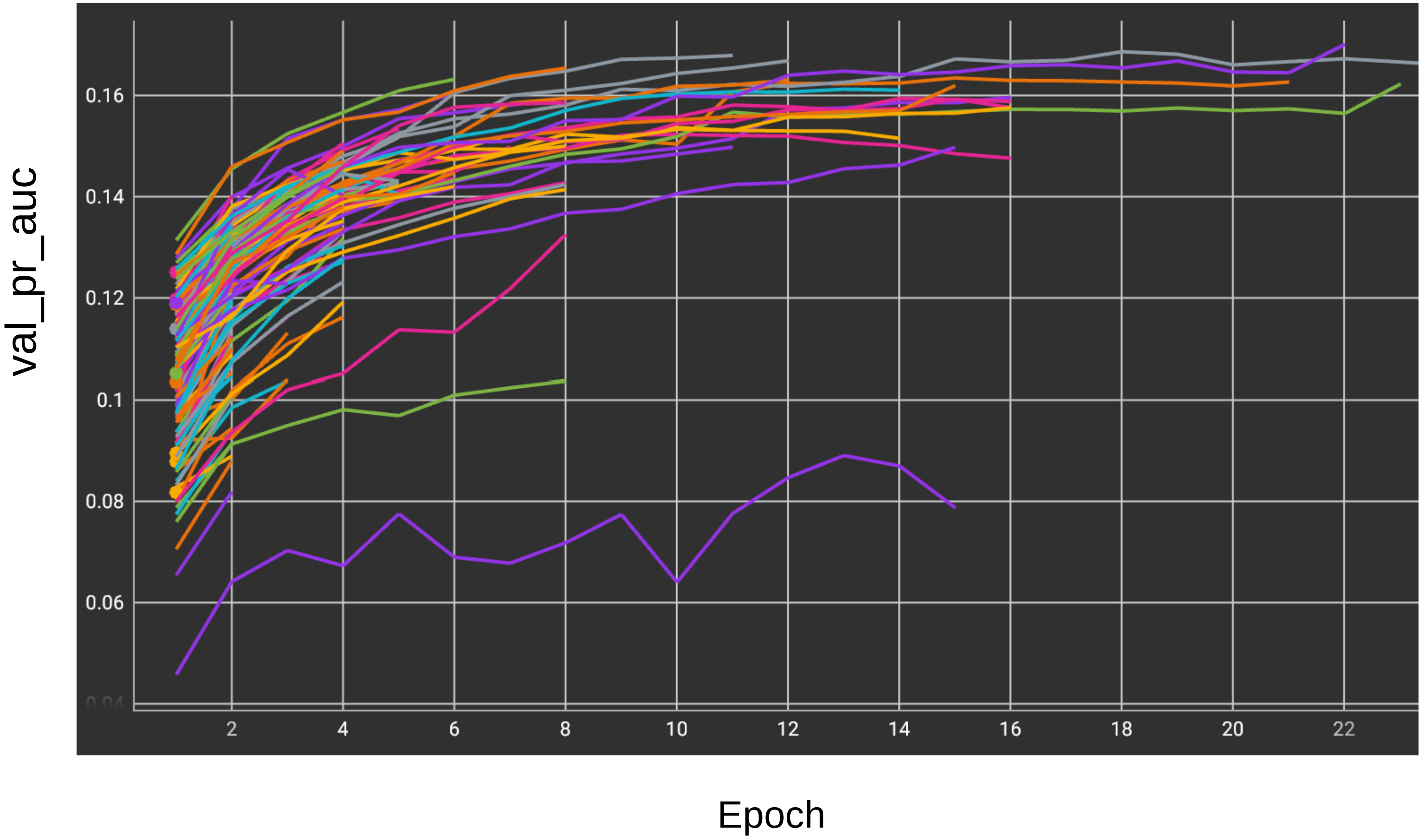
Second AHPO

- I went back to the architecture from manual HPO since it performed better
- Hyperparameter ranges are not anymore manually chosen such that they have at maximum 9 million params
- Instead, I implemented that configurations with too less or too many params are immediately terminated, resulting in 0.3 e6 < params < 9 e6
- Similar hyperparameters tuned as before, with the addition of tuning the parameters of the Adam optimizer

```python
class ParamSpace:
    def __init__(self) -> None:
        self.learning_rate = tune.loguniform(1e-3, 1e-2)
        self.decay_rate = tune.loguniform(1e-3, 1e-1)
        self.batchsize = tune.choice([16, 32])
        self.shortcut = tune.choice([True, False])
        self.batchnorm = tune.choice([True, False])
        self.dropout_rate = tune.loguniform(1e-3, 5e-1)
        self.kNN = tune.choice([25, 30, 35, 40])
        self.number_kernel_network_nodes_begin = tune.choice([32, 64, 128, 256])
        self.number_kernel_units = tune.choice([2, 3, 4])
        self.number_edgeconvs = tune.choice([3, 4, 5, 6])
        self.number_constant_dense_layers = tune.choice([1, 2, 3])
        self.number_decreasing_dense_layers = tune.sample_from(
            lambda spec: np.random.randint(
                0,
                spec.config.number_edgeconvs,
            )
        )
        self.exponent_basis = tune.choice([1.9, 1.95, 2, 2.05])
        self.beta_1 = tune.choice([0.88, 0.89, 0.9, 0.91, 0.92])
        self.beta_2 = tune.choice([0.9988, 0.9989, 0.999, 0.9991, 0.9992])
        self.epsilon = tune.choice([0.01, 0.1, 1.0])
```

Second AHPO

- Second AHPO is currently running
- Results look much more promising
- Analysis of the relevance of different hyperparameters is in preparation
- In total, I got granted 50k GPU hours from the NHR@FAU in Erlangen
- About 1/3 of that is used so far



Conclusion

- **Summary:**
  - Ray Tune was used as a framework for an automatic hyperparameter optimization
  - First AHPO used EdgeConv blocks with exponentially increasing and decreasing MLPs
  - Models did not reach performance of manual HPO: maybe a bottleneck?
  - Second AHPO is currently ongoing with variations of the architecture from the manual HPO
  - Results look more promising

- Ideas what could have gone wrong in the first AHPO?
- Suggestions which architectures I could try out in the next weeks?

Thank you very much for your attention!