Learning to reinforcement learn for Neural Architecture Search *

Jorge Gómez Robles

* M.Sc. project supervised by dr. ir. Joaquin Vanschoren

Department of Mathematics and Computer Science

The Neural Architecture Search problem

Neural Networks for Image Classification

Dataset → Neural Network

CIFAR-10 → DenseNet
ResNet

Arbitrary dataset → ?

Neural Architecture Search (NAS)

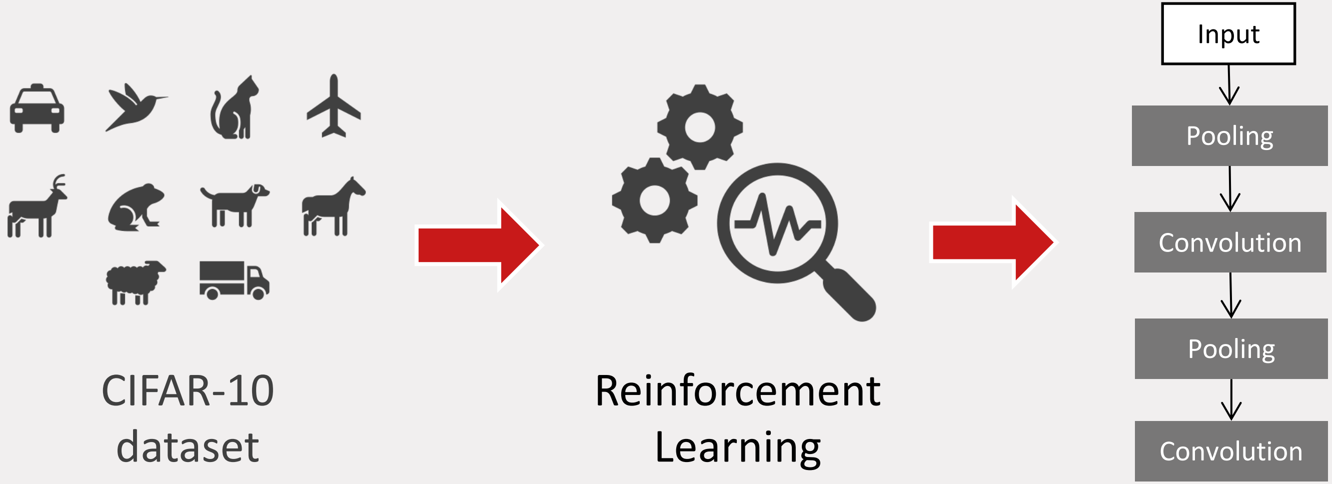
Goal

- Automate the creation of neural architectures for **any** dataset of interest.

Popular methods

- Bayesian optimization
- Evolutionary algorithms
- Reinforcement learning (RL)

The current NAS with RL framework



Flaws of the current NAS framework

- The run is very expensive
  - 28 days with 500 GPUs (B. Zoph and Q. V Le, “Neural Architecture Search with Reinforcement Learning”, 2017)
  - 3 days with 32 GPUs (Z. Zhong *et al.*, “BlockQNN: Efficient Block-wise Neural Network Architecture Generation”, 2018)
- For every new environment (i.e., dataset), a RL run must be performed from scratch.

The research question

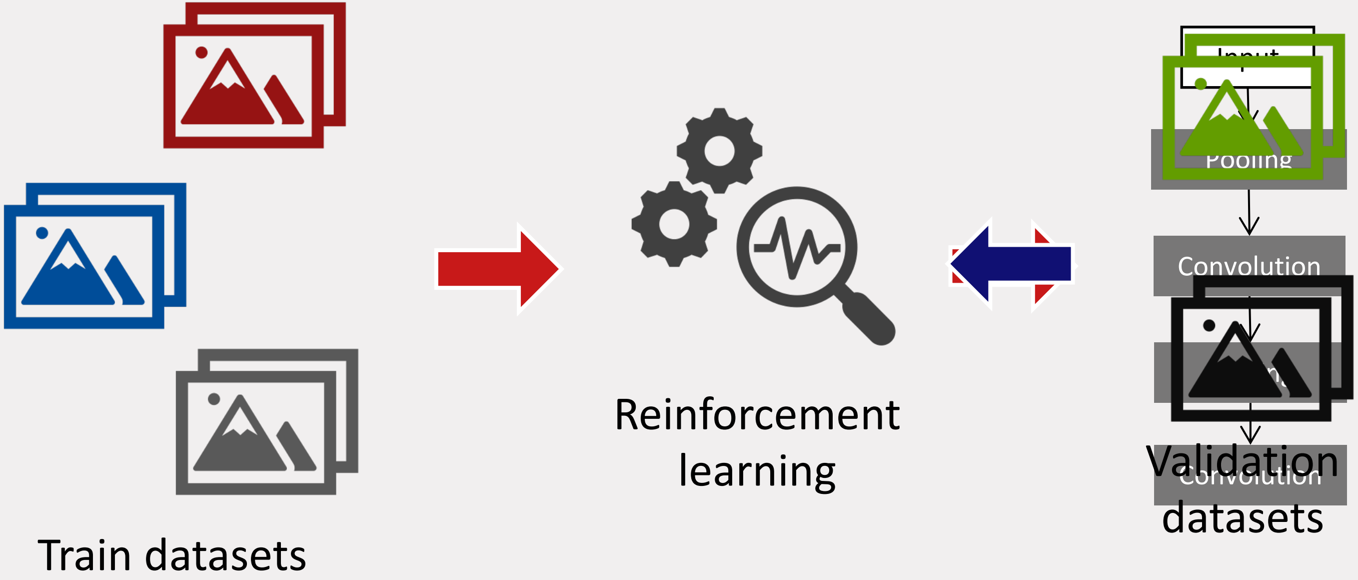
Research question

- Is it possible to learn an **adaptive strategy (policy)** to design CNNs for image classification so that we can transfer it to avoid **training from scratch**?

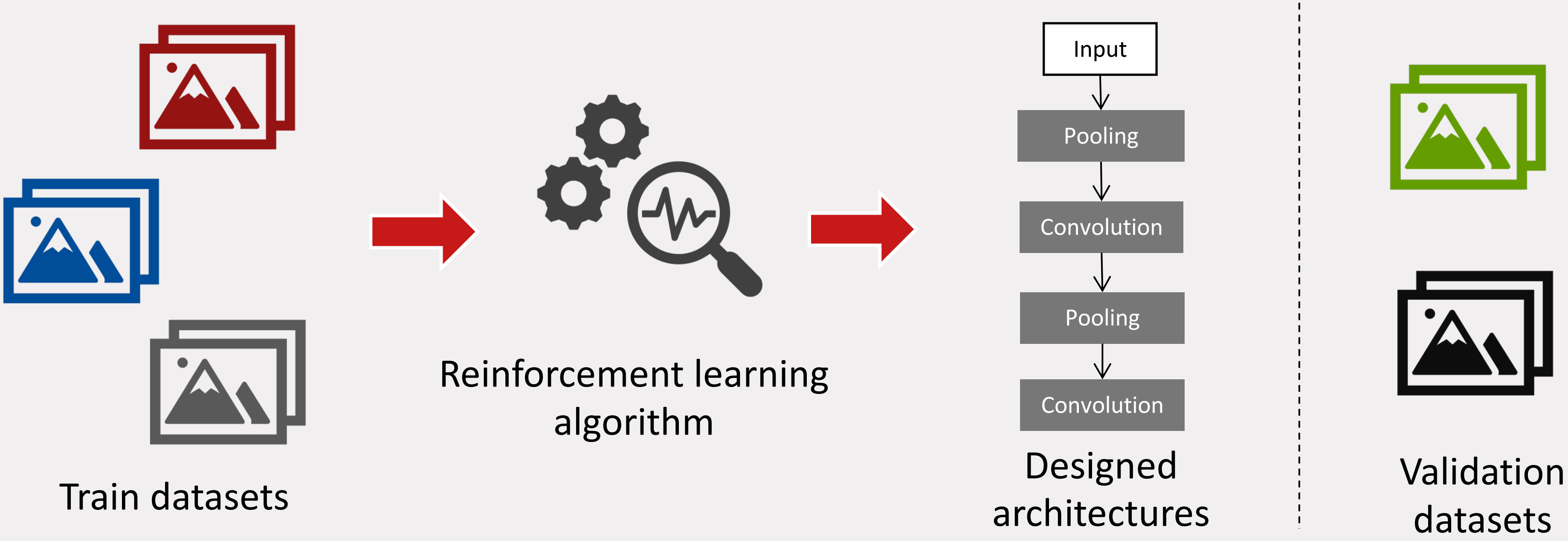
Consequences:

1. Transfer the policy and use it as an initialization to train on new environments
2. Transfer the policy to new environments without training

Proposed solution


Pooling
Convolution
Convolution
Train datasets
Reinforcement learning
Validation datasets

Analyzing the components thoroughly



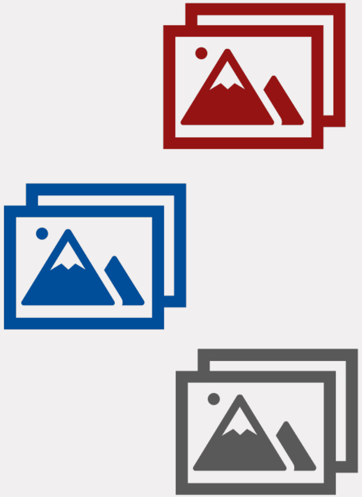
Analyzing the components thoroughly

Reinforcement learning

Use a **meta-reinforcement learning algorithm:**

- J. X. Wang et al., "*Learning to reinforcement learn*", 2016.
- Build on top of a the A2C algorithm
- An LSTM-based meta-learner that learns the relation between all the agent-environment interactions within an episode.

Analyzing the components thoroughly



Datasets

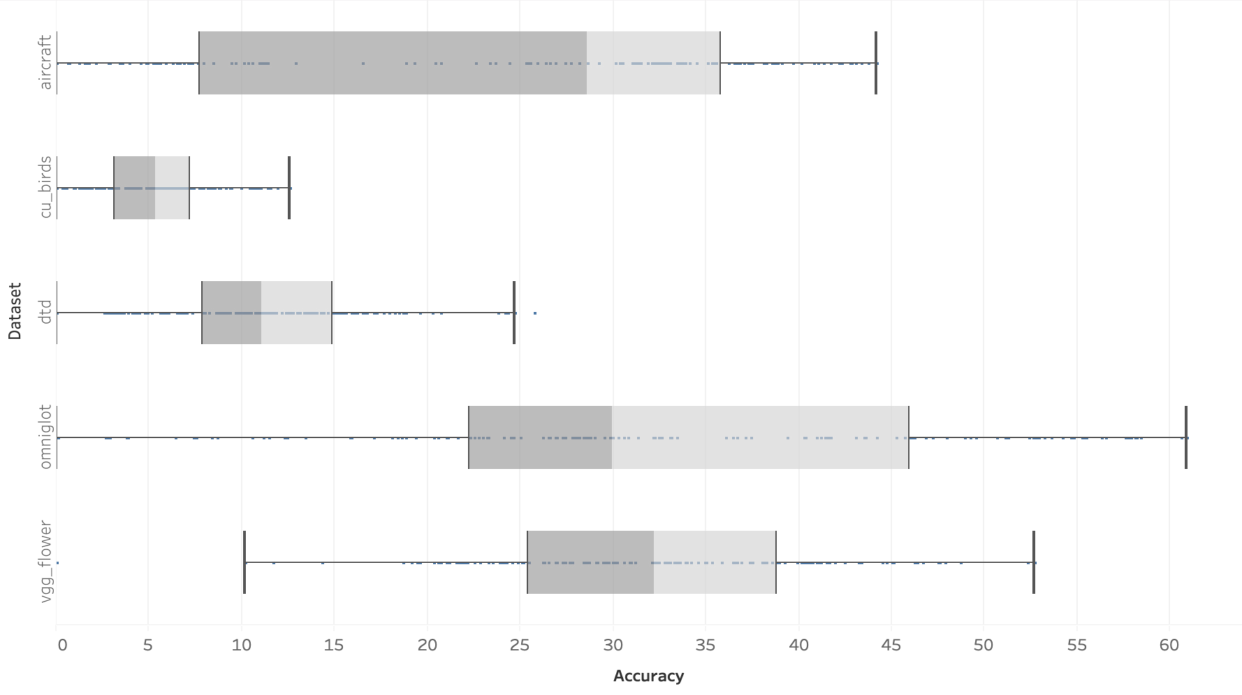
Five datasets from the the **meta-dataset**.

- E. Triantafillou et al., “*Meta-Dataset : A Dataset of Datasets for Learning to Learn from Few Examples*”, 2018.

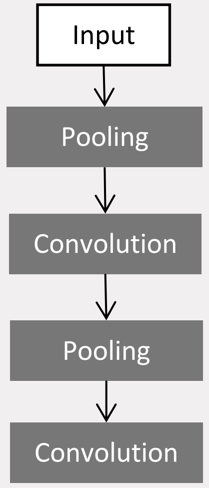
Dataset	ID	N Classes	N Obs	Usage
FGVC-Aircraft	aircraft	100	10,000	Validation
CUB-200-2011	cu_birds	200	11,788	Validation
Describable Textures	dtd	47	5,640	Training
VGG Flower	vgg_flower	102	8,189	Training
Omniglot	omniglot	1623	32,460	Training

Validation

Training



Analyzing the components thoroughly



Designed architectures

Based on the methodology of:

- Z. Zhong et al., "*Practical Block-wise Neural Network Architecture Generation*", 2017.

Highlights:

- 6 types of layers in the network
- Chain-structured networks only
- Maximum number of layers: 10
- Early-stop training to avoid long runs (12 epochs)

TU/e

Type of layer	Hyper-parameter
Convolution	Kernel size = {1, 3, 5}
MaxPooling	Pool size = {2, 3}
AvgPooling	Pool size = {2, 3}
Terminal	-

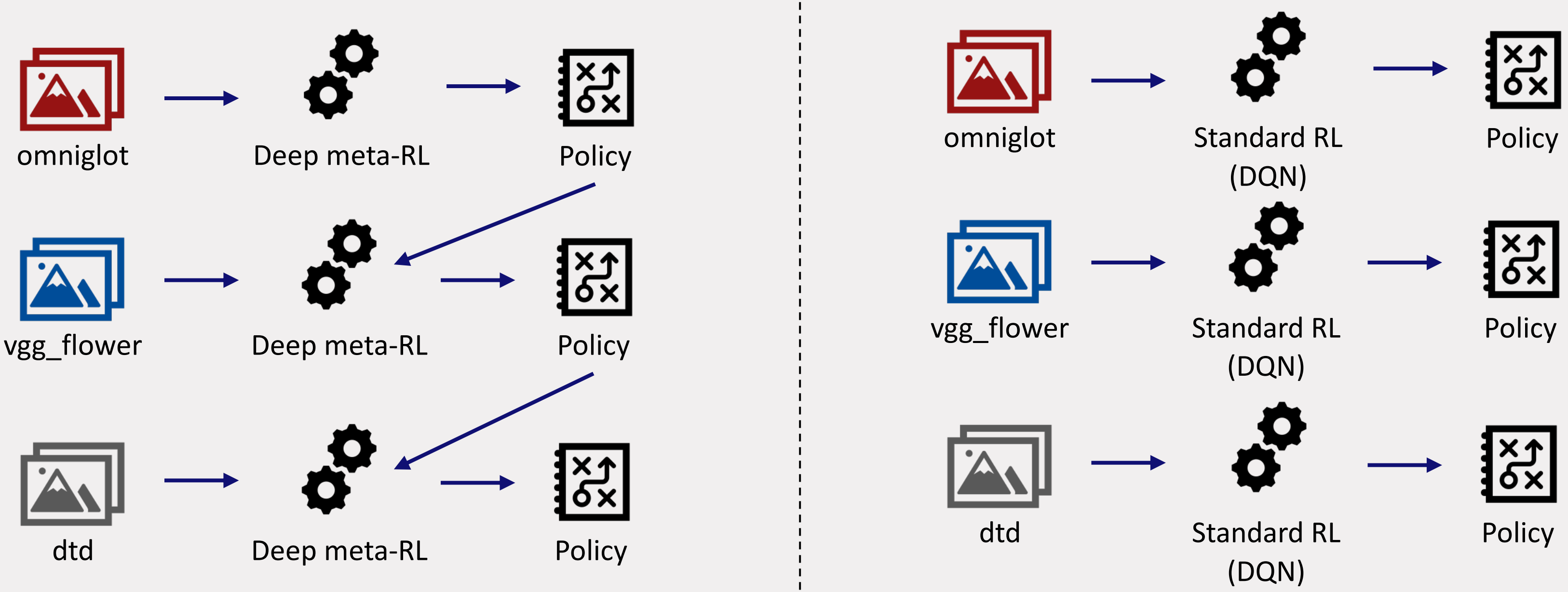
Software

NasGym
github.com/gomerudo/nas-env

Deep meta-RL
github.com/gomerudo/openai-baselines

Experiments and results

Experiment 1: training with deep meta-RL

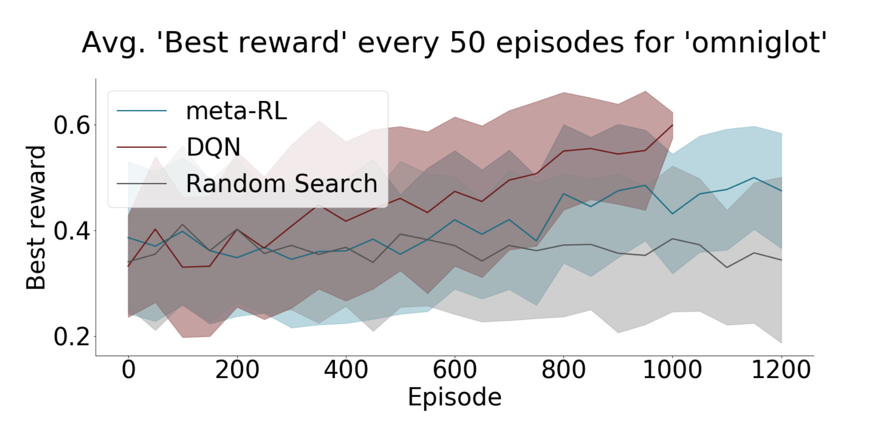


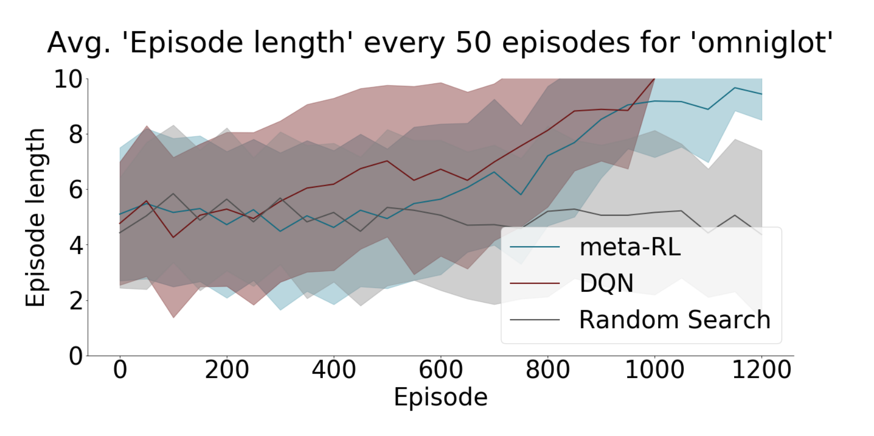
TU/e

Goal:

- Observe if deep meta-reinforcement learning gives an advantage over standard RL
  - *Best reward* -> Quality (the best designed network)
  - *Episode length* -> Complexity (episode length equal to the depth of the network)
- Study if the agent can adapt
  - *Policy entropy* -> exploration
  - *Proportion of actions* -> strategy per environment

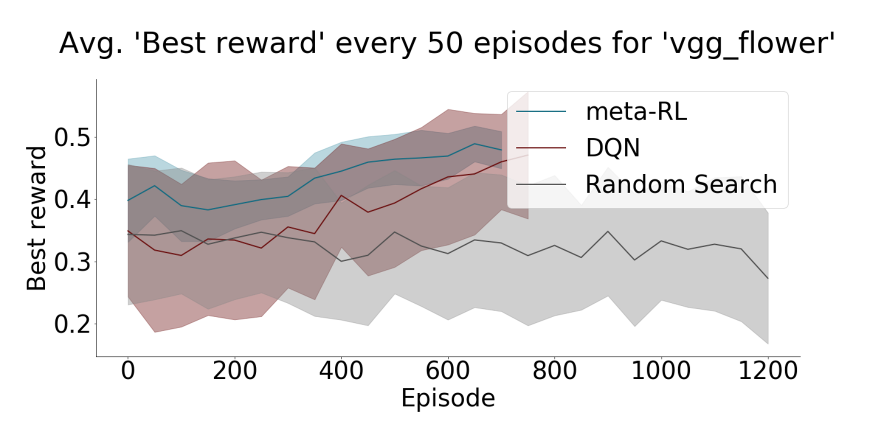
Results of Experiment 1: omniglot

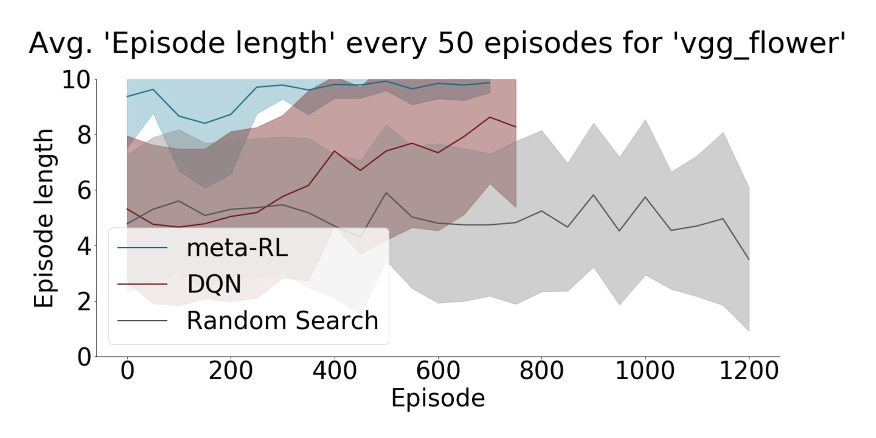




* Confidence range is 1 STD

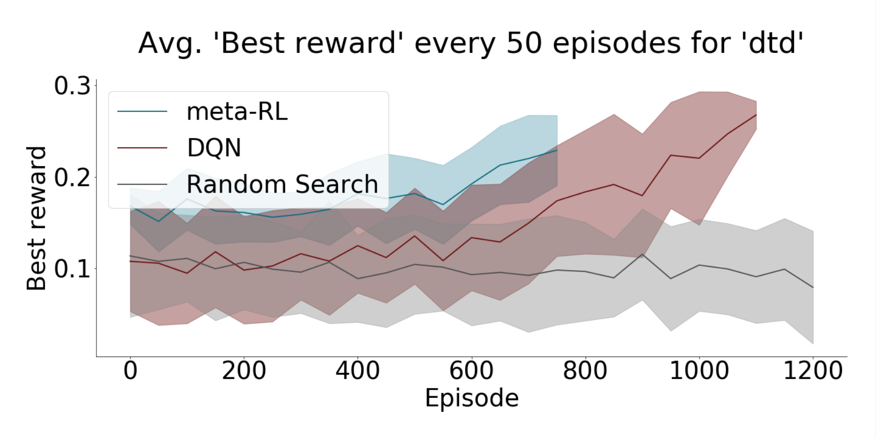
Results of Experiment 1: vgg_flower

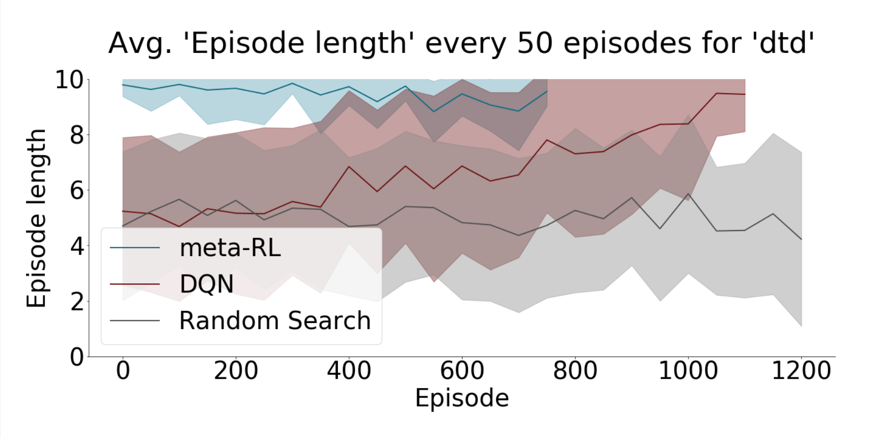




* Confidence range is 1 STD

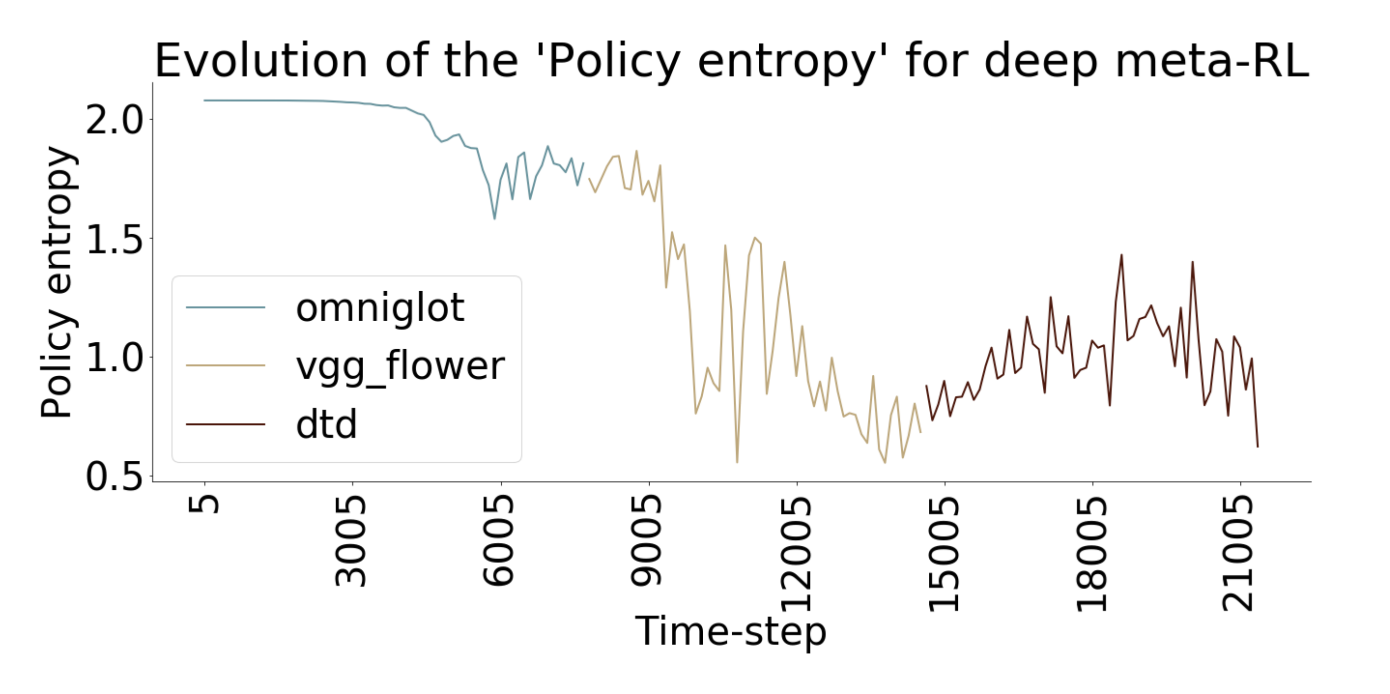
Results of Experiment 1: dtd



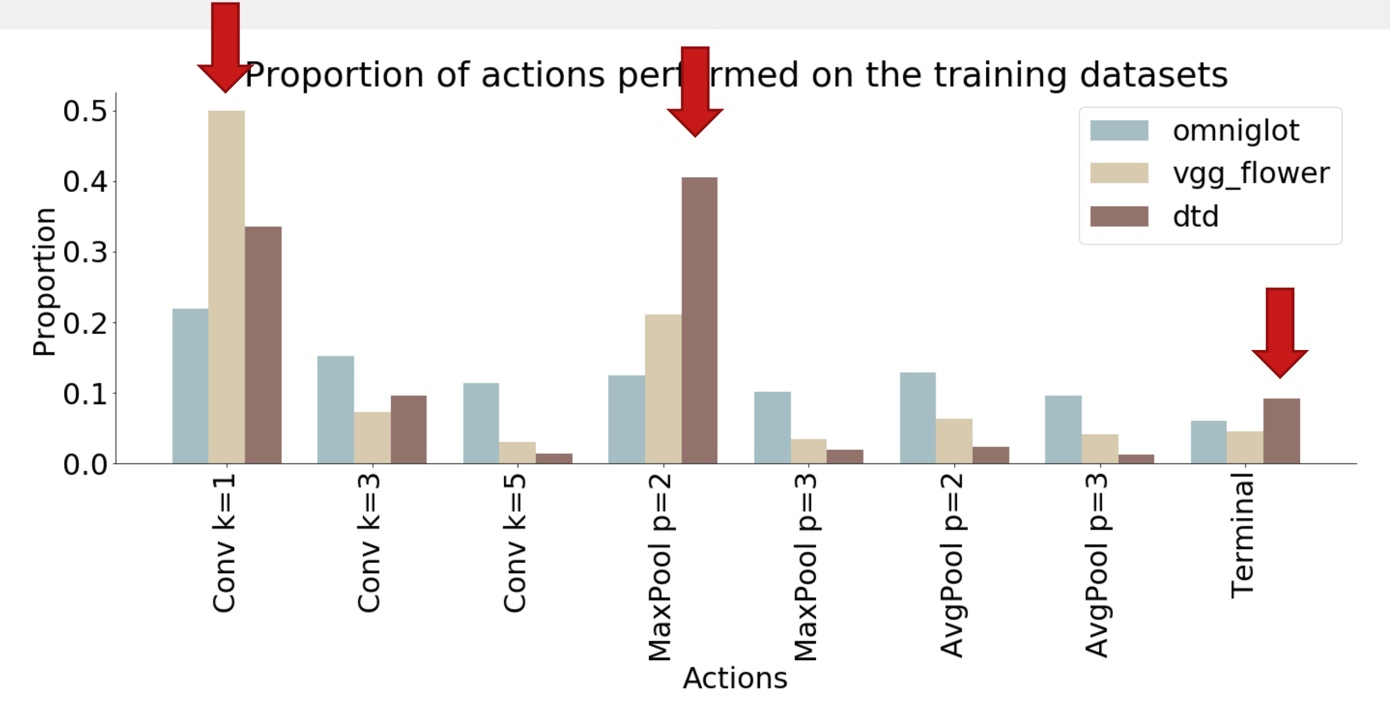


* Confidence range is 1 STD

Results of Experiment 1: exploration per environment

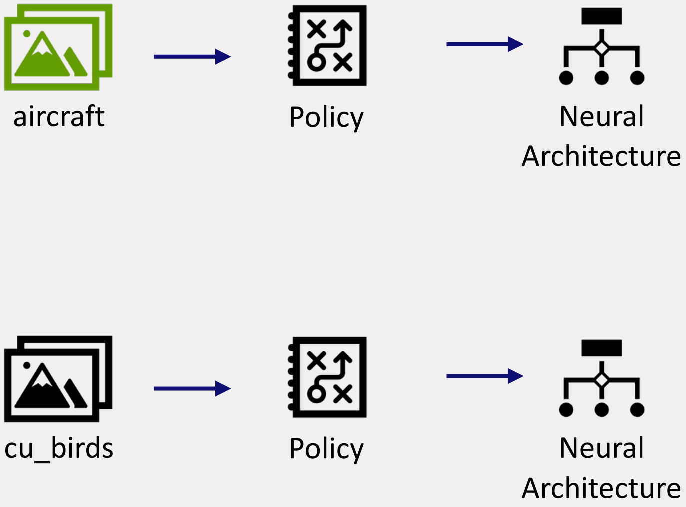


Results of Experiment 1: the strategy



TU/e

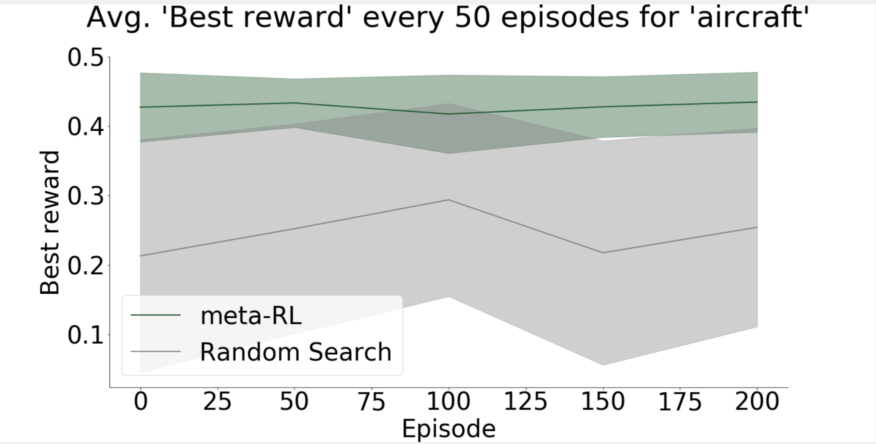
Experiment 2: evaluating the policy (Part A)

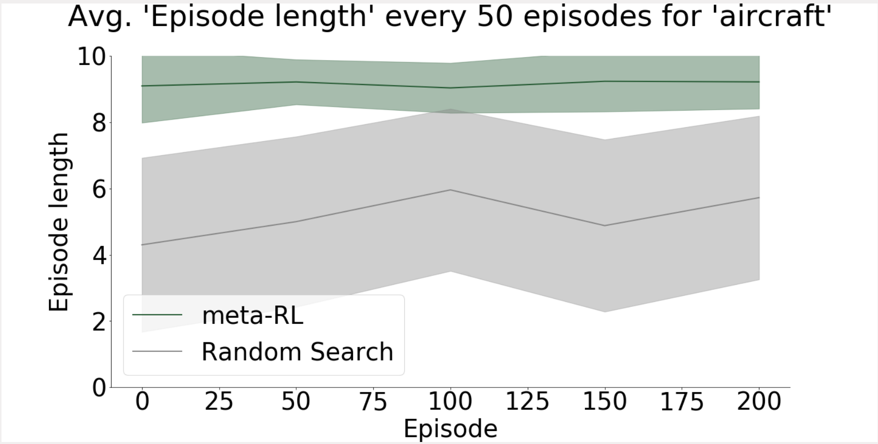


Goal:

- Study the strategies deployed by the agent

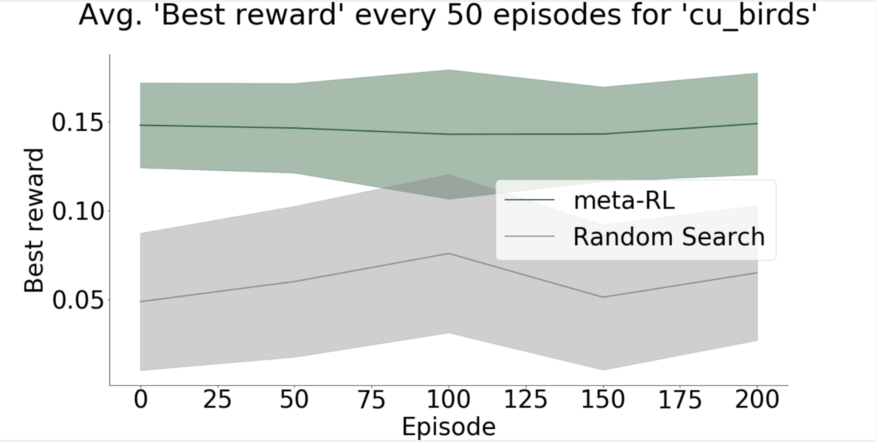
Results of Experiment 2: aircraft

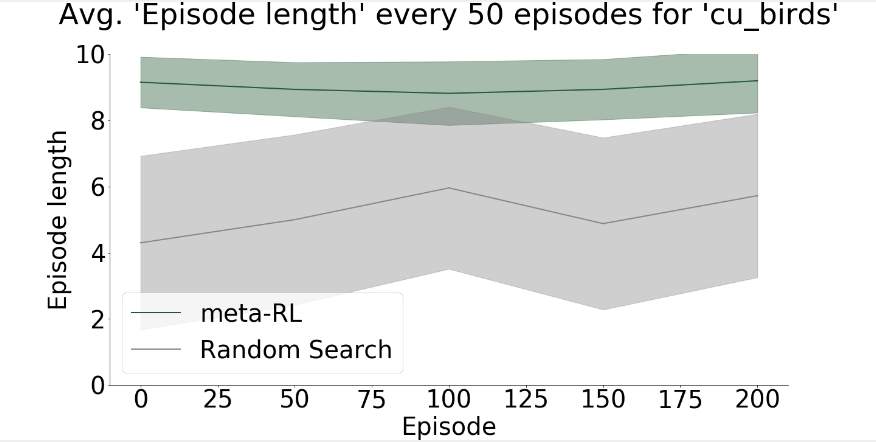




* Confidence range is 1 STD

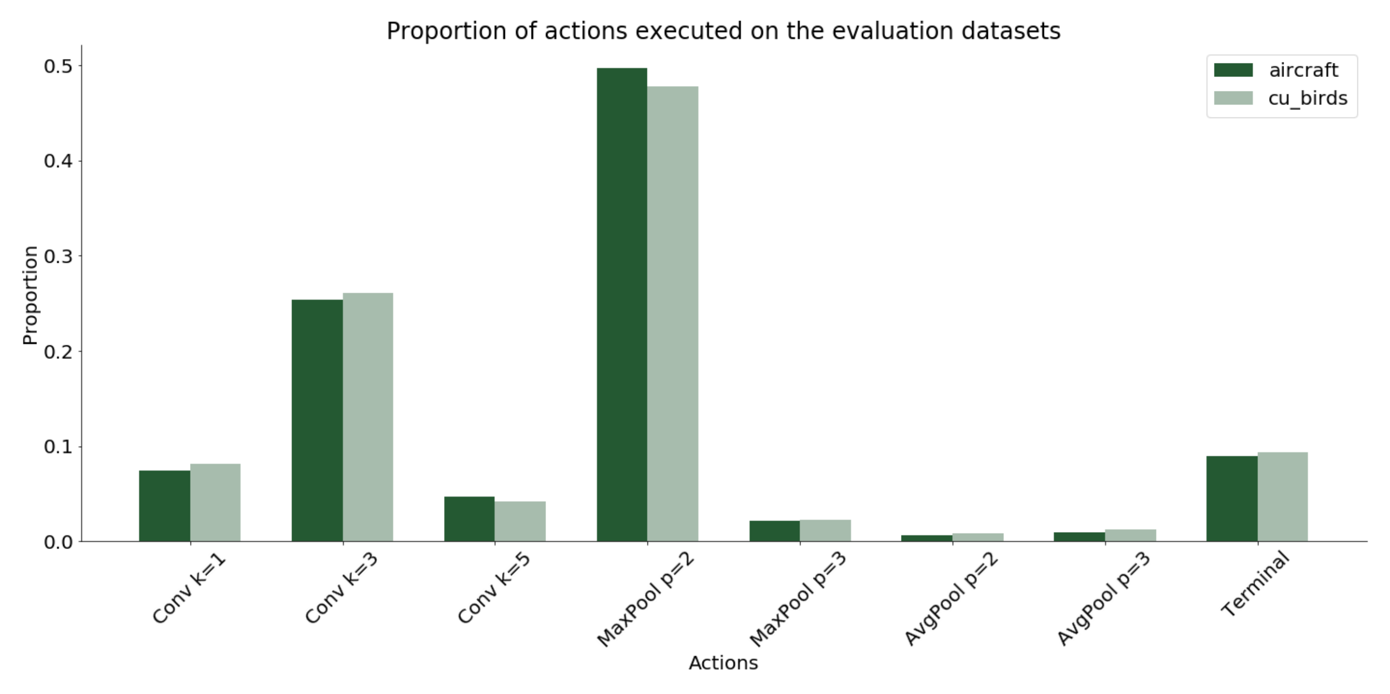
Results of Experiment 2: cu_birds





* Confidence range is 1 STD

Results of Experiment 2: the strategy



Experiment 2: evaluating the policy (Part B)

aircraft

Best 2 Neural Architectures

cu_birds

Best 2 Neural Architectures

Intensive Training (100 epochs)

Shorthened VGG19

Intensive Training (100 epochs)

Dataset	Deep meta-RL (1st)	Deep meta-RL (2nd)	Shortened VGG19
aircraft	49.18 ± 1.20	**50.11 ± 1.02**	30.85 ± 10.82
cu_birds	23.97 ± 1.28	**24.24 ± 0.90**	6.66 ± 1.98

Conclusions

Conclusions

- Deep meta-RL shows a good behavior when the policy is transferred during training
  - Shows indications of adaptation between environments
  - Outperforms standard RL
  - Shows consistency (small variance)
- During the evaluation the behavior can potentially be improved
  - The strategies deployed on different environments are not *ad-hoc*
  - The agent can design networks that outperform manually engineered solutions

What can we do better?

- Hyper-parameter tuning of deep meta-RL
  - Encourage exploration
  - Make learning faster
- Increasing the number of time-steps for the agent-environment interaction during training
  - Might help to improve the results during evaluation

Thanks!

Back-up slides

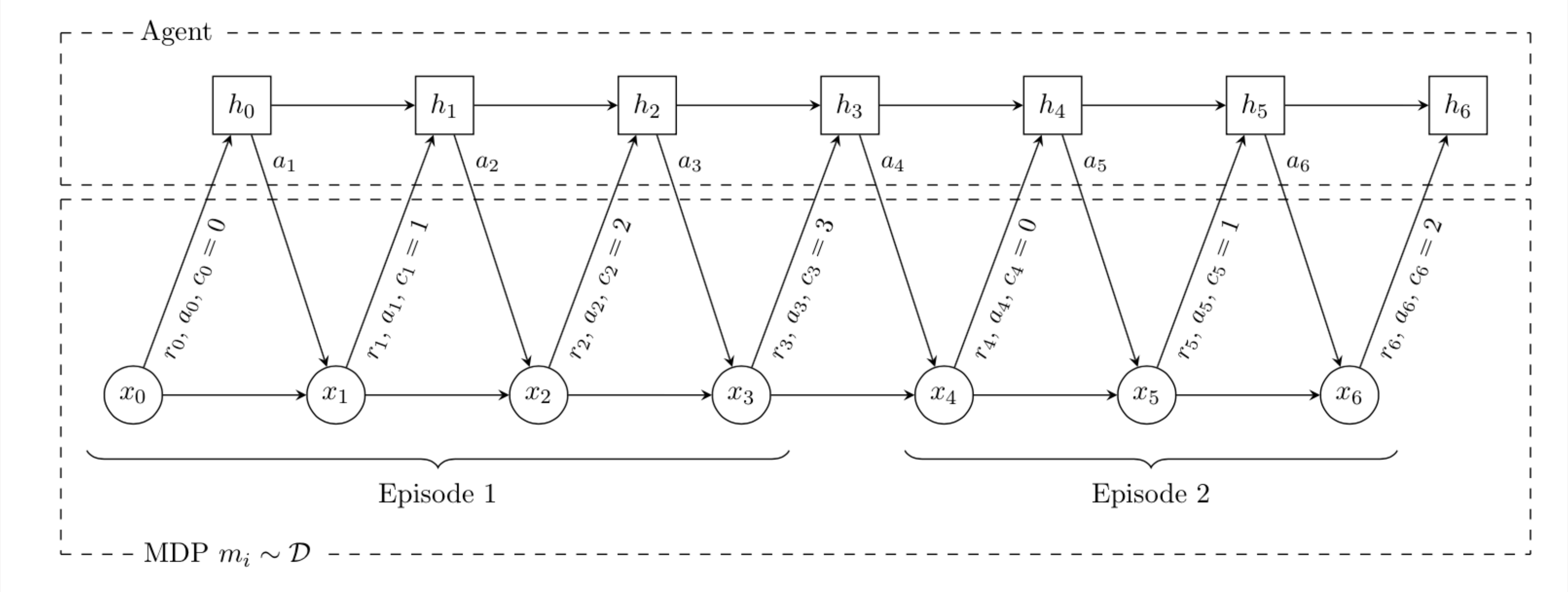
Results of Experiment 1: running times

Environment (dataset)	Deep meta-RL	DQN
omniglog	11 days 9h	6 days 14h
vgg_flower	7 days 23h	5 days 15h
dtd	6 days 17h	6 days 4h
Total	**26 days 1h**	**18 days 9h**

Results of Experiment 2: running times

Environment (dataset)	Time
aircraft	2 days 6h
cu_birds	2 days 22h

The deep meta-RL interaction



The A2C architecture

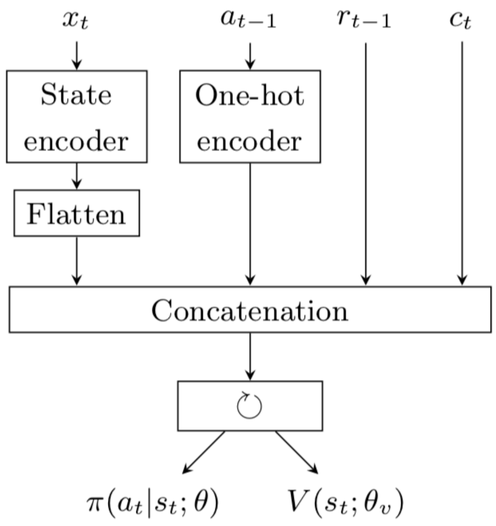


Figure 7: Illustration of the *meta*-A2C architecture. In our implementation, the "State encoder" follows the procedure explained in Section 4.2.1, and the recurrent layer is an LSTM with 128 units.

The policy optimization function

Formally, let t be the current time step, $s_t = x_t \cdot a_{t-1} \cdot r_{t-1} \cdot c_t$ a concatenation of inputs, $\pi(a_t|s_t;\theta)$ the policy, $V(s_t;\theta_v)$ the value function, H the entropy, $j \in \mathbb{N}$ the horizon, $\gamma \in (0,1]$ the discount factor, η the regularization coefficient, and $R_t = \sum_{i=0}^{j-1} \gamma^i r_{t+i}$ the total accumulated return from time step t. The gradient of the objective function is:

$$\nabla_\theta \log \pi(a_t|s_t;\theta) \underbrace{(R_t - V(s_t;\theta_v))}_{\text{Advantage estimate}} + \underbrace{\eta \nabla_\theta H(\pi(s_t;\theta))}_{\text{Entropy regularization}} \quad (1)$$

Prediction module

- Dense layer 1024 units
- ReLU
- Dropout 0.4
- Dense layer n_classes

Training

- Learning rate: 0.001
  - Reduced by a factor of 0.2 every 5 epochs
- Beta1 = 0.9
- Beta2 = 0.999
- Epsilon= 10e-8

Time-steps for experiment 1

Environment	Deep meta-RL	DQN
omniglot	8000	6500
vgg_flower	7000	5500
dtd	7000	7000

Hyper-parameters for experiment 1

Environment	A2C	DQN	Training of networks
d = 10 (max layers) $\tau = 10$ (Maximum length of episode)	j = 5 (number of steps before updating the network) $\gamma = 0.9$ (discount factor) $\eta = 0.01$ (Entropy regularization) $\alpha = 0.001$ (Learning rate)	Buffer size = t_max / 2 Target's model batch size = 20 ε: Linear decay from 1 to 0.1 $\alpha = 0.0005$ (Learning rate)	Batch size = 128 Num. epochs = 12