Towards Explaining Actions of Learning Agents

Bruno Rodrigues
NOVA LINCS, NOVA School of Science and Technology
Caparica, Portugal
bac.rodrigues@campus.fct.unl.pt

Matthias Knorr
NOVA LINCS, NOVA School of Science and Technology
Caparica, Portugal
mkn@fct.unl.pt

Ludwig Krippahl
NOVA LINCS
Caparica, Portugal
ludi@fct.unl.pt

Ricardo Gonçalves
NOVA LINCS, NOVA School of Science and Technology
Caparica, Portugal
rjrg@fct.unl.pt

ABSTRACT

Agents increasingly use Deep Neural Networks to process sensor information and make decisions. While these models have been shown to provide excellent results, they come with the disadvantage that they behave like black boxes, mapping inputs to outputs in a way that is hard for humans to understand. This is a serious disadvantage because it makes it harder to predict how agents will act in unexpected situations, which is especially dangerous when agents have to interact physically with humans, such as self-driving vehicles or industrial robots, but also creates risks for agents such as chat bots and other virtual agents since their actions may result in legal liabilities or reputation damage. Being able to explain decisions taken by these neural networks that guide the agents is important for preventing incorrect behavior and for building trust and providing legal justifications whenever necessary. This applies not only to interactions with humans, but also to multiagent systems. In this paper, we build on a recent framework on Explainable AI that uses small neural networks to map activations from a trained deep neural network to relevant concepts in a logical formalization of the domain, which in turn can be used to provide explanations for the outputs of the original network. Since this framework is applied to the deep neural network at inference time, after training, it can be applied to neural networks used in agents regardless of whether these were trained using supervised or reinforcement learning. We show that a potential bottleneck of the approach, the creation of such mapping networks, can be solved by employing automated neural architecture search. This paves the way towards applying this approach to more advanced use cases of explaining decisions of agents based on deep neural networks, regardless of how these networks were trained.

KEYWORDS

Explanations, Neural Architecture, Reinforcement Learning

1 INTRODUCTION

Deep neural networks (DNNs) are currently being used to solve a variety of problems, such as web search [33], image [35] and video [14] classification, recommendation algorithms for social media websites [5], finance [9] and decision support systems with a large impact on humans, such as loan granting [39], job recruitment [6] and university application processing [40]. They are also being used to guide agents that interact with humans, such as virtual assistants [44] and autonomous vehicles [43].

The potential impact of DNNs on human lives demands guarantees of quality, safety and fairness from DNN-powered systems. Unfortunately, DNNs are opaque, black-box, models that provide results with no intelligible indication of why that output was generated. This is a problem when the models are used to inform human decisions, but even more so when the models are used without real-time human supervision, as for autonomous agents, from chat bots to self-driving vehicles. Explainability, in a broad sense the ability to make a model's decision understandable to humans, helps mitigate this problem by giving us additional means of evaluating the model. Black-box models can be evaluated by performance measures on known examples, but without an insight into how the model operates, such measures only correlate indirectly with whether the model is working correctly as intended. Explainability is recognized as an important concern whenever using DNNs has important consequences. E.g., in healthcare, explainability of diagnosis systems increases trust in deciding treatment routes [17], and physicians rank explainability as the most desirable feature for a clinical decision support system [17, 37].

These concerns motivate the rising importance of the field of Explainable AI (XAI), which focuses on developing AI systems whose behavior can be understood by humans [8]. There is a wide range of approaches to state-of-the-art XAI solutions. For example, some are model-agnostic, such as those using proxy models [30]; others focus on the relevance of different parts of the input more relevant to the model's decision [26]; others explain some instances by relying on other similar cases [15], among many other alternatives. A detailed overview of XAI falls outside the scope of this paper (for a recent review see [20]), but it is relevant to point out the complexity of the problem of finding explanations, which causes this diversity.

Among these approaches, there is a novel method, called concept mapping [7], that finds correspondences between activation patterns in a DNN and concepts in an ontology for providing a formal justification for the output of the DNN. The core idea is that a DNN performing some task will rely on internal representations of relevant features, which can be mapped to a formal representation of human knowledge about the domain if this knowledge captures the same relevant features. Thus, by providing a logical description of the domain formalized as an ontology, humans can specify those concepts and relations that can be used to build acceptable explanations. This is important because good explanations strongly depend on the application domain, the purpose of the explanation,

Proc. of the Adaptive and Learning Agents Workshop (ALA 2023), Cruz, Hayes, Wang, Yates (eds.), May 29-30, 2023, London, UK, https://alaworkshop2023.github.io/. 2023.

and its target audience. Once these concepts and their relations are specified, one can find a mapping between the activations of some neurons inside the trained DNN and these concepts in the ontology. This, in turn, allows one to provide an explanation for the observed classification utilizing off-the-shelf reasoners that can infer non-trivial knowledge from ontologies, including justifications for a model's behavior. In addition, the explanations produced by a reasoner can easily be translated into natural language, allowing laypersons to directly understand a produced justification without an expert's help.

Employing additional neural classifier models – the mapping networks – which are trained to map the sub-symbolic internal representations of the main network onto one relevant symbolic concept, one can achieve excellent results in creating such explanations using this formalism [7]. This approach can be applied to any trained DNN, and thus can be useful for learning agents, as long as they rely on artificial neural networks. This is important because autonomous agents that operate outside human control but must interact with humans are an extreme example of a system that requires a good understanding of the models so that it is possible to trust such agents.

However, in order to apply this method, it is necessary to create mapping networks that can accurately find a correspondence between neuron activations in the DNN and the appropriate concepts. In realistic problems, this is not trivial and poses a challenge to the application of this method, since a mapping network that produces an inaccurate labeling will then lead to incorrect justifications.

In this paper, we argue that the approach of mapping networks, since it is applicable to DNN after training, can also be applied to DNN trained with reinforcement learning, as it is often the case for training autonomous agents. We show with a number of extensive experiments using a controlled, synthetic, dataset how Neural Architecture Search (NAS) can be used to automate the process of finding mapping networks, which are the central pieces of the method for extracting symbolic meaning from subsymbolic representations, and also demonstrate the application to a simple reinforcement learning example. The main objective of this paper is to provide a method for generating explanations for the behavior of agents controlled by DNN.

The remainder of the paper is structured as follows. We start by providing the necessary background in Section 2. Then, in Section 3 we review the concept mapping method for justifying a neural network's output, followed by a demonstration of its application on a neural agent in Section 4. In Section 5, we describe neural architecture search, the framework used in this paper. Section 6 features a discussion on the extensive empirical tests showing that neural architecture search can be used to provide good concept mapping models with minimal human effort. We summarize our findings in Section 7, and discuss possible future work.

2 BACKGROUND

This section provides a brief overview of relevant notions and notation necessary for understanding the following material.

Deep neural networks (DNN) are artificial neural networks that stack several layers of neurons. The neuron is the basic building block of such networks, and consists in a linear combination of inputs followed by a nonlinear funcion. By stacking such layers the DNN can transform input vectors in ways that better identify informative patterns. The neuron weights are adjusted by back-propagation in order to optimize a loss function that captures the desired purpose of the DNN (e.g. cross-entropy for classification, mean squared error for regression, and so forth).

By combining different types of neurons and operations, DNN can be adapted to different problems. One important example are Convolutional Neural Networks (CNNs) [25], which excel at Computer Vision tasks such as image classification and segmentation. These DNN use convolutional layers, where the same set of neuron weights are applied to regularly-spaced small patches of the image to produce feature maps. This allows the identification of useful patterns in the image while reducing the total number of parameters in the network. A typical CNN classifier consists of a stack of convolutional layers to extract features from the image input followed by fully connected layers that output a vector with the probabilities of the input belonging to each class.

Reinforcement Learning. DNNs are trained by comparing the network output to a desired output and minimizing the loss function. In supervised learning these desired outputs are given by a training set consisting of previously labelled examples. In Reinforcement Learning (RL) [36] the examples can also be generated by having the agent explore the decision space and using feedback from the environment. Deep Q-Learning [21], a variant of Q-Learning [42], is an example. In this type of learning, a DNN learns to approximate Q-function, a maping from each state-action pair to the discounted reward, which can be used to determine the best action to take at each state by selecting the action with the highest Q-value. Using a DNN makes it possible to use high-dimensional and unstructured inputs such as images.

Description Logics (DLs). DLs [2] are fragments of first-order logic whose reasoning tasks are usually decidable. Here, we briefly recall the standard DL $\mathcal{ALC}$. The basic elements to represent knowledge in DLs are: *individuals* that represent objects in a domain of discourse; *concepts* that group together individuals with common properties; and *roles* that relate individuals. The sets N_I, N_C, and N_R of individual names, concept names and role names, respectively, form the basis to construct the syntactic elements of $\mathcal{ALC}$ according to the following grammar (in which $A \in N_C$ and $r \in N_R$):

$$C \quad \longrightarrow \quad \bot \mid \top \mid A \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \exists r.C \mid \forall r.C$$

Intuitively, the logical operators can be read as negation ($\neg$), conjunction ($\sqcap$), and disjunction ($\sqcup$) of concepts, the existence of a relation r from an individual belonging to the class to one belonging to C ($\exists r.C$), and for all relations r from an individual belonging to the class, the related individual belongs to C ($\forall r.C$) (cf. [2] for the formal semantics). An *ontology* then contains *axioms*, namely *assertions* of the form $C(a)$ and $r(a_1, a_2)$, that assign an individual a to a concept C and relate two individuals a_1, a_2 by role r, respectively, as well as *concept inclusions* of the form $C_1 \sqsubseteq C_2$ to state subsumption of concept C_1 by concept C_2 and *concept equivalence* $C_1 \equiv C_2$ as a shortcut for $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$.

Given such an ontology, one can perform standard reasoning tasks, e.g., whether the specification admits a model, i.e., is consistent, which can be used to validate the specification, or whether

a certain formula is an implicit logical consequence of the given specification. The latter also allows us to determine which axioms are used to determine a certain logical consequence, which provides a justification for the obtained inference [11].

3 CONCEPT MAPPING

In this section, we briefly recall material on producing justifications for a neural network's output via concept mapping [7].

The objective is to justify the outputs of a trained neural network employing reasoning over a formal specification of the task's domain, given the output of the neural network and the presence of certain characteristic features somewhere in the network's model, which are relevant for inferring the obtained output. This requires a formal specification of the task's domain, in the form of an ontology, containing concepts equivalent to the ones extracted by the neural network and those relevant for inferring the main concepts.

Building on the assumption that if a concept is indeed relevant to the task, it is present in the network's internal representations, mapping networks are trained on those representations. To do so, first a subset of the dataset is labeled with respect to these relevant concepts, and the mapping networks are trained to generalize from it. In other words, the mapping networks are trained to learn what internal behavior a concept triggers in the main network when it appears in the input. For their outputs to be useful in producing symbolic explanations for the main network's behaviour, the mapping networks own behaviour need not be similarly explained. We are merely interested in predicting with high confidence whether a concept is present.

Not all of the main network's activations need be fed to the mapping network. After all, conventionally-sized CNNs can have tens of millions of trainable parameters, and given that all the data is fed forward through the network, we should expect the same information to be found at multiple levels. CNNs excel at computer vision due to how the stacking of multiple convolutions allows for simple patterns (e.g. contrast, lines, basic shapes) to be detected in the first layers and more complex ones (e.g. ears, tail, wheels) to be extracted in the latter ones. At the end of the convolutional part of a CNN, the relevant features have been identified and largely abstracted from their position in the input. Therefore, the representations of the dense part of the network are viable candidates for input for the mapping task.

With this in place, it is possible to produce justifications for the main network's output for a given example by a) providing the main network with the example and having it produce a classification; b) using the mapping networks to identify the concepts that the main network found in that example; and c) using the suitable DL reasoner to produce a justification for the network's output based on the ontology and the identified concepts.

Fig. 1 contains a graphical representation of the method using the example of classification of images of trains [7] based on the characteristics of these trains - an example image is given on the bottom-left. Such images are classified by the main network, i.e., the model whose behaviour we want to explain, e.g., classifying the example image in Fig. 1 as *TypeB*. Taking (part of) the inner activations of the main network as input, the mapping networks are trained to detect certain relevant concepts, previously determined

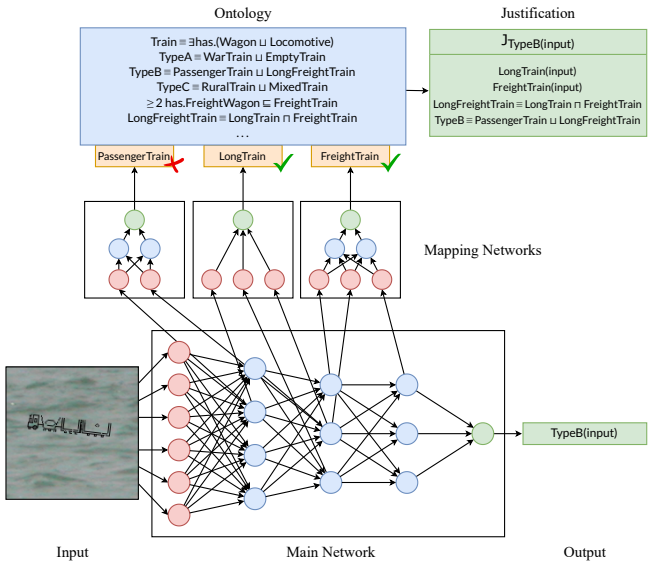

Figure 1: Overview of the Concept Mapping Method [7]

within an ontology for the domain, taking into account input images that are labelled accordingly. Once trained, an image when being classified by the main network also gets assigned which concepts are detected by the mapping networks. In this example, the mapping networks detected the concepts *LongTrain* and *FreightTrain* in the image. The ontology for the domain, an excerpt of which is shown on the upper left,[1] together with the observed concepts for the image allow to infer *TypeB*. The justification given on the right contains the two observations and two axioms from the ontology. The resulting justification can be read as we infer *TypeB*, because *TypeB* is a *PassengerTrain* or a *LongFreightTrain* (line four), and the latter is equivalent to *LongTrain* and *FreightTrain* (line three), which corresponds to the observations.

The cost of deploying this method depends on the complexity of the ontology, which is influenced by the complexity of the main task and the desired explanatory power. More complex problems will increase the number of relevant concepts to be detected, which in turn requires the training and usage of more mapping networks. The usage of neural architecture search employed in our work (see Sects. 5 and 6) may help address this problem.

4 USING CONCEPT MAPPING TO EXPLAIN AN AGENT'S BEHAVIOR

In this section, we show how the method, described in the previous section, can be applied to explain the behavior of a DNN-powered agent, such as one trained with DQN.

To demonstrate this, we employed concept mapping on a DQN agent trained to play tic-tac-toe. In this game, two players, X and O, take turns marking the spaces on a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical,

[1] Here, also qualified number restrictions are used, allowing one to describe objects that have at least two *FreightWagons*, but that does not affect the method presented here as long as the DL reasoner used for inference is capable of reasoning with those.

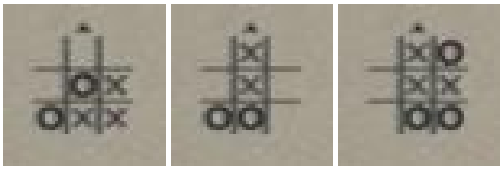

Figure 2: Sample images from the tic-tac-toe dataset

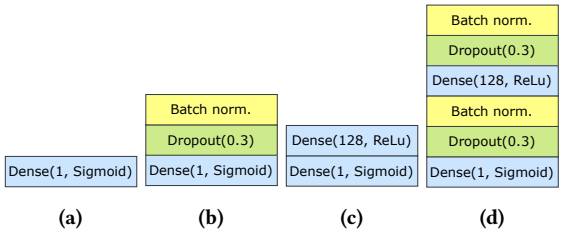

Figure 3: Suite of tested mapping architectures

or diagonal row wins the game. The agent receives information about the game state via an image depicting the board, extracts relevant features from the image and uses them to make a decision that maximizes its chances of winning the match.

Given tic-tac-toe's simplicity, it is possible to have a dataset containing images of every legal board. Given three potential states per grid cell - an X, a circle, or nothing - there exist $3^9 = 19683$ ways that the 3x3 grid can be filled in, 5478 of which can actually occur in a game. For this demonstration, we generated 50x50 pixel images using Blender's Python API [10]; depicting a top-down view of a board outlined in dark gray, on a textured background (Fig. 2).

The task involves extracting information from image data, therefore the network powering the agent is a CNN, which, after we trained the agent against a random player for 10,000 games, learned to predict the expected reward of placing an X on each available position, given solely an image of the current board.

To explain the trained agent's behaviour using concept mapping, we need to be able to extract relevant concepts from its internal activations. Let us consider concepts referring to whether a certain position contains a certain symbol. For example, we can number the board positions as follows:

0	1	2
3	4	5
6	7	8

Then we can denote by $O4$ the concept *"There is an O in position 4"*, and so forth for the other symbol and positions. This can then be used with an ontology characterizing reasons for each of the possible moves, containing axioms such as $X3 \sqcap X4 \sqsubseteq X5$ and $O2 \sqcap O8 \sqsubseteq X5$ (assuming that X is placed by the learning agent).

We tested four models on extracting the eighteen resulting concepts, whose architecture is shown in Fig. 3 [2].

[2]"Dense(n,a)" denotes a fully connected layer with n neurons followed by an a activation function. "Dropout(p)" denotes a *dropout* layer with an probability p of ignoring a given neuron's output

Model	Avg BCE Loss	Avg Top-1 Acc.
(a)	2.71×10^{-2}	0.9994
(b)	2.41×10^{-4}	1
(c)	1.93×10^{-2}	0.9998
(d)	1.58×10^{-4}	1

Table 1: Average Binary Cross Entropy Loss and Top-1 Accuracy across eighteen concepts.

For every step in a game, the agent receives an image depicting the current board and chooses the legal action that maximizes the predicted Q-value [3]. The next state is generated, and, after the opponent plays, a tuple of the type (s_t, a, r, s_{t+1}, d) is stored in the agents memory bank, where s_t contains the initial state, a the action taken by the agent, r the reward received as a consequence of that action, s_{t+1} the state produced by the action, and d a flag indicating whether s_{t+1} is a final state. These memories are used to learn the effect of certain actions in certain scenarios and to adjust the target Q-values being learned by the agent. At each step, the agent is trained on random memories from its bank.

The results (Table 1) show that the concepts can easily be extracted with excellent performance by all models, demonstrating the viability of concept mapping as an explanatory method for neural agents.

While these results show that the method is also applicable in this setting, at least from the point of view of finding such mapping networks this is not too surprising. In fact, the method supposes a trained main network, but how such network was trained is not important. However, common to both the examples in Sects. 3 and 4, is that the main network's task is rather simple, which allows even simple mapping network architectures found manually to perform well.

Before we proceed showing how the application of this methodology to larger problems can be aided by determining mapping networks in an automated fashion using neural architecture search, we first recall some necessary notions in the next section.

5 NEURAL ARCHITECTURE SEARCH

The field of Neural Architecture Search (NAS) [28] stems from an interest in streamlining and automating the process of neural architecture design, which requires substantial human effort and expertise. NAS systems employ machine learning techniques to learn neural architectures that minimize the validation-set loss after training on a given dataset. These are its main components:

- **Search Space:** The domain of architectures that are considered as candidates.
- **Search Strategy:** The method by which the Search Space is traversed. Common search strategies employed by early NAS systems are random search (RS) [32], reinforcement learning (RL) [3, 45], evolutionary algorithms (EA) [27], bayesian optimization (BO) [4], monte carlo tree search (MCTS) [24], and sequential model-based optimization (SMBO) [12].

[3]Initially, to promote exploration, there is a high probability that the agent will instead perform a random action. This probability decays to zero as training progresses.

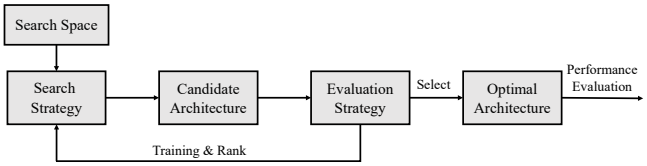

Figure 4: The general framework of NAS [28]

- **Evaluation Strategy:** The method used to benchmark the candidate architectures. The search strategy relies on this protocol to provide, at least, a comparative ranking between different candidates to inform the search direction.

In sum, a NAS system traverses its search space using its search strategy, assessing different candidates with its evaluation strategy. Fig. 4 shows a diagram of this general framework.

In a performance comparison [28] of NAS methods on the CIFAR-10 dataset, NAS-RL [45], achieved a 3.65 error rate after 22400 GPU days, while DARTS [19], a newer system that employs several optimization techniques, achieved 2.76 error rate after 4 GPU days. With its state-of-the-art results and diminished computational requirements, DARTS has the desired attributes of a NAS system for the concept mapping application, which is why we briefly review it next.

5.1 DARTS

One of the ways DARTS achieves its high efficiency is by using a modular search space, meaning that rather than designing a whole architecture, it designs a computational cell that can be stacked to build a final architecture. A cell is represented as a Directed Acyclic Graph (DAG), in which each node $x^{(i)}$ is an internal representation of the input and each edge (i, j) is an operation $o^{(i,j)}$ that further transforms $x^{(i)}$. The content of each node $x^{(j)}$ is computed based on all of its incoming edges:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \tag{1}$$

Learning the optimal cell consists of learning the optimal operation to place on each of the edges. In addition to other candidate operations, a *Zero* operation is used to indicate a non-existing connection between two nodes.

The key optimization of DARTS is to continuously relax the NAS problem, and thus enable the use of efficient gradient descent algorithms. To achieve this relaxation, each edge of the cell represents, during training, a *Softmax* over all candidate operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x) \tag{2}$$

where $\mathcal{O}$ is the set of all candidate operations and α a set of vectors $\alpha^{(i,j)}$ containing mixing weights associated to each operation of each edge. Going further, we will refer to the set of these mixing weights α as the arch weights - since they encode the architecture - and to the trainable parameters of the candidate operations as the model weights.

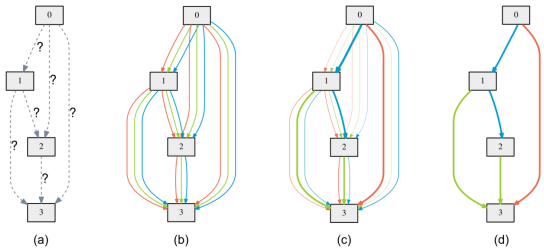

Figure 5: An overview of DARTS [19].
(a) The problem, find the optimal cell. (b) Continuous relaxation of the problem by replacing each edge with a set of mixed operations. (c) Joint optimization of architecture and network weights. (d) Derivation of final architecture by replacing mixed operations with the learned optimal operation.

Once the optimal cell has been learned, the discrete architecture can be obtained by replacing the mixed operations with the learned (i.e. most likely) operation:

$$o^{(i,j)} = argmax_{o \in \mathcal{O}} \quad \alpha_o^{(i,j)} \tag{3}$$

In each step of the architecture search, the arch weights α are first optimized by descending the validation-set loss. Then the model weights are optimized by descending the training-set loss of the architecture found in the upper-level optimization. This process is repeated until convergence, at which point the final architecture is derived. Figure 5 contains a summary of the DARTS framework.

6 DESIGNING CONCEPT MAPPING MODELS WITH NEURAL ARCHITECTURE SEARCH

To empirically assess the merits of NAS for concept mapping, we implemented an adaption of DARTS, geared towards learning of mapping architectures (henceforth referred to as DARTS-CM [4]). First, we changed the set of candidate operations. In the original paper, a series of convolutional candidate operations were used for the learning of CNNs, which does not apply to concept mapping. Therefore, we used the following operations: [5]

- Skip Connect (i.e. identity function)
- Batch Normalization [13]
- Layer Normalization [1]
- Dropout [34]
- Gaussian Dropout [29]
- ReLU [22]
- LeakyReLU [38]
- Dense (i.e. fully connected layer)

We further changed the cell structure to be linear (i.e. each node only outputs to the next).

To allow for robust testing of our DARTS implementation, we developed Arch-MAX (Architecture: Modular, Artificial, Explainable), a set of image classification datasets with identical features that differ only in how difficult the images are to classify. The images

[4]The implementation is provided at https://github.com/brunoacr/DARTS-CM
[5]Note that we exclude the *Zero* operation used in the original work. Subsequent DARTS-based works [18, 31, 41] have noted how the operation hinders stability, and we had similar observations.

(a) Commercial (b) Residential (c) Industrial

Figure 6: Samples from the Arch-MAX dataset

	1	2	3	4	5	6
Models	✓	✓	✓	✓	✓	✓
Textures		✓	✓	✓	✓	✓
Camera - Small variance			✓	✓	✓	✓
Background elements				✓	✓	✓
Camera - Large variance					✓	✓
Contrast						✓
Brightness						✓

Table 2: Variations present in each of Arch-MAX'S complexity levels

contain depictions of buildings that vary in their surroundings and visual attributes. The building itself may contain a door, windows, awnings, billboards, porches, wall signs, roof statues, tiled roofs, chimneys, and pipes. Surrounding the building, there may be eating areas, pole signs, vending machines, construction machinery, trucks, and cars. Excluding the tiled roof and roof statues, the features can appear in one of multiple locations, and windows and awnings can also appear in varying quantities. Different combinations of these features are mapped different classes of houses, which in turn belong to one of three main classes - Residential, Commercial, and Industrial (see Fig. 6). These mappings are modeled in an ontology, which defines the main concepts as a disjunction of different lower-level concepts representing the various kinds of buildings, for example:

$$\text{Residential} \equiv \mathit{CountryHouse} \sqcup \mathit{MiscResidential} \sqcup \mathit{Suburban}$$

"Residential Buildings are at leas one of these: (...), and all of them are residential buildings"

Then each of the lower-level concepts is defined with respect to which features appear on the building:

$$\mathit{CountryHouse} \equiv \exists \mathit{has}.\mathit{Car} \sqcap \exists \mathit{has}.\mathit{TiledRoof}$$

"Any Building that features a Car and a TiledRoof is a CountryHouse (and vice-versa)"

The ontology is a parameter of the Arch-MAX generator[6]. The user only needs to provide an ontology that encodes what combinations of features describe each class, and the generator will produce an equal amount of examples for each class, according to the provided "recipes".

Multiple methods are used to make the data harder to classify: alternating the model used for a given feature; varying textures; varying the camera's position; adding background elements; and randomizing the contrast and brightness of the image. Table 2 shows the six levels of complexity used for our experiments.

We used eighteen fully trained binary classification convolutional neural networks - one for each (*complexity*, *class*) pair - to evaluate our implementation of DARTS. The template shown in Fig. 7 represents the architectures that were used, where c and d are variables that determine the number of times the respective block is stacked. Convolutional layers in the same block share the same amount of filters, which increases from block to block with the network's depth. The first Convolutional block always has thirty-two filters, and that number doubles for each subsequent one. The

[6]At https://github.com/brunoacr/ArchMAX-Generator are provided generator and ontology, and the datasets at https://huggingface.co/datasets/bruno-cotrim/arch-max.

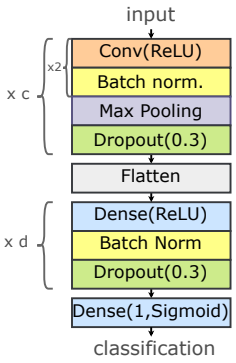

Figure 7: Convolutional Architecture Template used for the main networks.

Dense layer that directly precedes the output layer is set to have sixteen neurons, and every preceding layer has double the amount of its successor.

The models for complexities one to three were trained with eight thousand examples in the training set and around twenty-seven hundred in each of the validation and test sets. The models for complexities four to six were trained with twelve thousand examples in the training set and four thousand examples in each of the validation and test sets. Table 3 shows, for each main network, the values for c and d as well as the obtained loss and accuracy on the test set.

For each of the three main classes, we selected three relevant secondary concepts, and used DARTS to design architectures for the task of extracting each concept. The result of the architecture search is an encoding of a computational cell featuring four selected operations (See Fig. 8 for an example).

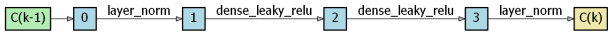

Figure 8: Learned cell for complexity 2, main class Industrial, concept MiscIndustrial

Complexity	Class	c	d	Loss	Acc
1	Commercial	3	2	0.036	0.9904
	Industrial	3	2	0.026	0.992
	Residential	3	2	0.051	0.9865
2	Commercial	3	2	0.0526	0.9814
	Industrial	3	2	0.055	0.9823
	Residential	3	2	0.0362	0.9883
3	Commercial	6	2	0.0721	0.9805
	Industrial	3	2	0.0973	0.9761
	Residential	3	2	0.0879	0.9817
4	Commercial	2	3	0.143	0.958
	Industrial	2	3	0.0743	0.9773
	Residential	2	3	0.0778	0.9827
5	Commercial	6	2	0.1545	0.9453
	Industrial	6	2	0.1325	0.955
	Residential	6	2	0.1209	0.9562
6	Commercial	6	2	0.1487	0.9456
	Industrial	6	2	0.0953	0.9741
	Residential	6	2	0.1193	0.9591

Table 3: Test-set performance of each main network.

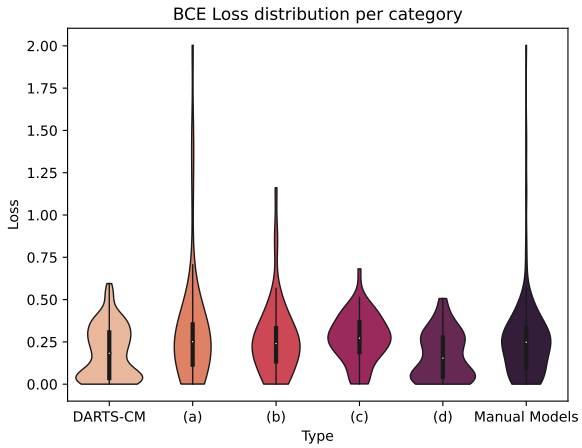

Figure 9: BCE Loss distribution per category.

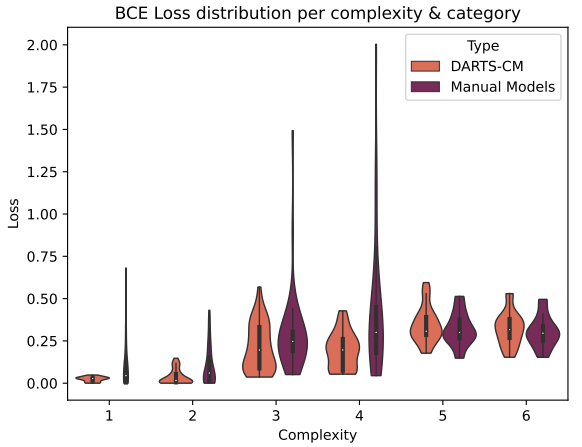

Figure 10: BCE Loss distribution per complexity and category.

After the architecture search, we build, train and evaluate models with one to four copies of the learned cell. To provide a frame of reference, we also evaluated the four architectures from Fig. 3 on the same tasks. All mapping networks used 750 examples in the training set, 250 in the validation set, and 1000 in the test set. In all sets, exactly half of the examples contain the concept being learned. Each model was trained for a maximum of fifty epochs, stopping earlier if there is no improvement to the validation loss for fifteen epochs. The model uses a Binary Cross-entropy loss function and the Adam [16] Optimizer with a learning rate of 1e−3, momentum $\beta = (0.9, 0.999)$. All the values shown result from an averaging of five trials.

Figure 9 shows the distribution of the test-set binary-cross-entropy loss value obtained by the models in each category. The density of the values in a given range is given by the width of the graph in that range. The values shown include the results obtained by all of the architectures in each category prior to selecting the best performing model for each one. DARTS-CM exhibits lower variance in the loss values than the Manual Models, providing more consistent and predictably good performance in most cases.

Figure 10 shows the same Loss distribution, but now only for the DARTS-CM and aggregated Manual Models categories, and subdivided by complexity. This graph illustrates that the concept mapping task is highly sensitive to the main network's task. Even when performing the same classification task, changes in the complexity of the images affect the extraction of the same concepts. This further emphasizes the advantages brought by a system such as DARTS-CM, which shows a higher resilience to those changes than the manual models.

Notably, while there is a visible pattern for complexities one to four, complexities five and six appear to break that pattern, showing more homogeneous results, where both categories stay within a shorter and more similar range in terms of performance. This phenomenon likely owes itself to the fact that the main networks for complexities five and six did not learn the main task to the same degree as the ones for lower complexities (Table 3), effectively setting a bottleneck on concept-mapping performance, and reducing the performance differential caused by architecture quality. After all, a less accurate main network can reasonably be assumed to be similarly less accurate at predicting certain secondary features, resulting in situations where a concept appears in the input, but is not identified by the main network. Given that the concept mapping task is to predict the latter, but the training labels encode the former, this causes the training data for the mapping models to effectively include wrong labels. This added noise makes the data harder for the concept mappers to fit - increasing the lower bound on loss for those complexities - and thus also harder to over-fit - reducing the upper bound on loss - which results in the observed smaller range of loss values.

In Figure 11 we plot the distribution of the difference between the loss value of each architecture and the one from the best performing model for the same *(complexity, concept)* combination. High density

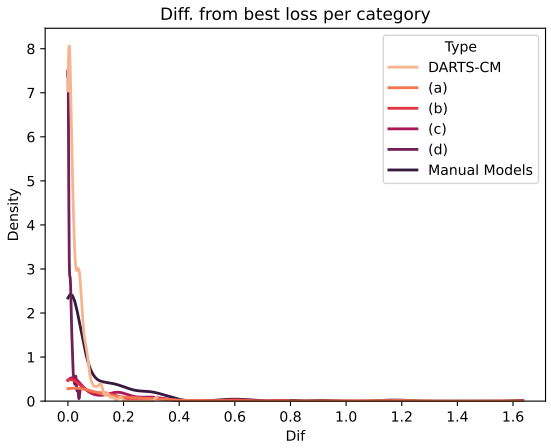

Figure 11: Distribution of difference in loss to the best performing architecture

near or at zero means that that category is often the best or close to the best. DARTS-CM exhibits a much higher density of values at or near zero and a smaller range of values than the manual architectures.

6.1 Discussion

We saw that the test set loss of all models tends to increase with the complexity of the dataset, meaning that concepts from more complex classification tasks are also more complex to map. While DARTS showed more resilience to those changes, we believe further improvement of the methodology can return an even more robust system. Firstly, we would increase the training time allocated to the algorithm. DARTS-CM was trained on a maximum of fifty epochs with an early stopping policy with a patience of twenty epochs. Since it optimizes the arch weights at the same time as the model weights, operations with fewer trainable parameters usually get better early performance and are thus favored, meaning that optimal operations with high parameter counts require more time to surface. For low complexities - and better trained main networks - DARTS-CM manages to fully optimize the architecture in the allocated time, but it may require more time for higher complexities. Furthermore, the fact that the majority of the candidate operations used were non-parametric also reduces the attention received by the others. Therefore, removing some of those operations from the search-space could allow for faster convergence in more complex datasets.

We ran a preliminary experiment to test this intuition by allocating two hundred epochs to training, with an early stopping policy with a patience of thirty epochs. The concept $\exists has.Porch$ was mapped from the main network for the complexity four dataset and main class *Residential*. Additionally, we shortened the search space by leaving only the skip_connect, dropout, batch normalization, and the dense operations. DARTS-CM had previously achieved a test-set loss value of 0.0546 for this same configuration, being surpassed by a manual architecture that achieved a loss value of 0.0262. After these changes, DARTS-CM managed to find an architecture that obtained a loss value of 0.0243, obtaining an improved result, better than any of the manual architectures.

7 CONCLUSIONS

In this work, we have shown that the approach of mapping networks can be successfully applied to generate explanations for the behavior of agents controlled by DNN, which helps increase trust in the decisions taken by an autonomous learning agent. In particular, we have empirically demonstrated the merits of employing neural architecture search in the design of the necessary concept mapping architectures, reducing the effort of creating the latter, and thus making the methodology more easily applicable to more complex settings. Notably, we have shown that our approach, DARTS-CM, is able to use its larger search space efficiently to produce consistently high quality concept mapping models, that are overall more reliable than trial-and-error across changes in the main network's task and in the concepts being extracted.

As producing explanations for relatively intricate domains can conceivably require the extraction of many concepts, the initial set-up cost of DARTS becomes obviously worthwhile, given the much higher cost of individually fine-tuning architectures for each concept. DARTS-CM is a significant step towards making concept mapping a viable explanatory method, to be deployed at scale on wide range of systems.

As possible future work, we may modify the set of candidate operations. One possible solution is to have composite candidate operations, with multiple neural operations each. For example, one candidate operation could be a batch normalization layer followed by a dense layer followed by a ReLU activation function. By keeping the skip-connect operation, the search algorithm would remain able to find simple designs, but the operations with high parameter counts would be awarded more attention by the algorithm. Another route would be to experiment with different cell structures, increasing the number of nodes, or making the cell non-linear, with each node receiving multiple inputs from preceding structures.

Additionally, recent related work [18, 31, 41] has pointed out possible improvements for DARTS, that could be implemented into DARTS-CM. Of special note, XNAS [23] (eXperts Neural Architecture Search) introduces a novel optimization method based on the theory of prediction with expert advice, which seems better suited for the selection task that is NAS, while keeping DARTS' continuous relaxation of the NAS problem. XNAS also enables the resurfacing of operations with high parameter counts that become effective later in training, and shows promising results, both in terms of performance and search costs.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. We acknowledge partial support by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT.IP, and B. Rodrigues and M. Knorr acknowledge partial support of "Project Sustainable Stone by Portugal - Valorization of Natural Stone for a digital, sustainable and qualified future, nº 40, proposal nº C644943391-00000051, co-financed by PRR - Recovery and Resilience Plan of the European Union (Next Generation EU)".

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. https://doi.org/10.48550/ARXIV.1607.06450

[2] Franz Baader (Ed.). 2003. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, Cambridge, UK ; New York.

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing Neural Network Architectures using Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=S1c2cvqee

[4] Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 2018. *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. https://proceedings.neurips.cc/paper/2018

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, Shilad Sen, Werner Geyer, Jill Freyne, and Pablo Castells (Eds.). ACM, 191–198. https://doi.org/10.1145/2959100.2959190

[6] Jeffrey Dastin. 2018. Amazon scraps secret AI recruiting tool that showed bias against women. https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G

[7] Manuel de Sousa Ribeiro and João Leite. 2021. Aligning Artificial Neural Networks and Ontologies towards Explainable AI. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 6 (May 2021), 4932–4940. https://ojs.aaai.org/index.php/AAAI/article/view/16626

[8] Derek Doran, Sarah Schulz, and Tarek R. Besold. 2017. What Does Explainable AI Really Mean? A New Conceptualization of Perspectives. arXiv:1710.00794 http://arxiv.org/abs/1710.00794

[9] Adam Fadlalla and Chien-Hua Lin. 2001. An Analysis of the Applications of Neural Networks in Finance. *Interfaces* 31, 4 (2001), 112–122. https://doi.org/10.1287/inte.31.4.112.9662

[10] Blender Foundation. [n.d.]. *Blender API Documentation*.

[11] Matthew Horridge. 2011. *Justification based explanation in ontologies*. Ph.D. Dissertation. University of Manchester, UK. http://www.manchester.ac.uk/escholar/uk-ac-man-scw:131699

[12] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization (Lecture Notes in Computer Science)*, Carlos A. Coello Coello (Ed.). Springer, Berlin, Heidelberg, 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

[13] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. http://arxiv.org/abs/1502.03167 arXiv:1502.03167 [cs].

[14] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-Scale Video Classification with Convolutional Neural Networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 1725–1732. https://doi.org/10.1109/CVPR.2014.223

[15] Mark T Keane and Eoin M Kenny. 2019. How case-based reasoning explains neural networks: A theoretical analysis of XAI using post-hoc explanation-by-example from a survey of ANN-CBR twin-systems. In *Case-Based Reasoning Research and Development: 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8–12, 2019, Proceedings 27*. Springer, 155–171.

[16] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. https://doi.org/10.48550/arXiv.1412.6980 arXiv:1412.6980 [cs].

[17] Jean-Baptiste Lamy, Booma Devi Sekar, Gilles Guézennec, Jacques Bouaud, and Brigitte Séroussi. 2019. Explainable artificial intelligence for breast cancer: A visual case-based reasoning approach. *Artif. Intell. Medicine* 94 (2019), 42–53. https://doi.org/10.1016/j.artmed.2019.01.001

[18] Chao Li, Jia Ning, Han Hu, and Kun He. 2022. Enhancing the Robustness, Efficiency, and Diversity of Differentiable Architecture Search. https://doi.org/10.48550/arXiv.2204.04681 arXiv:2204.04681

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. arXiv:1806.09055 http://arxiv.org/abs/1806.09055

[20] Dang Minh, H Xiang Wang, Y Fen Li, and Tan N Nguyen. 2022. Explainable artificial intelligence: a comprehensive review. *Artificial Intelligence Review* (2022), 1–66.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 http://arxiv.org/abs/1312.5602

[22] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, Johannes Fürnkranz and Thorsten Joachims (Eds.). Omnipress, 807–814. https://icml.cc/Conferences/2010/papers/432.pdf

[23] Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik-Manor. 2019. XNAS: Neural Architecture Search with Expert Advice. http://arxiv.org/abs/1906.08031 arXiv: 1906.08031.

[24] Renato Negrinho and Geoff Gordon. 2017. DeepArchitect: Automatically Designing and Training Deep Architectures. https://doi.org/10.48550/ARXIV.1704.08792

[25] Keiron O'Shea and Ryan Nash. 2015. An Introduction to Convolutional Neural Networks. arXiv:1511.08458 http://arxiv.org/abs/1511.08458

[26] Vitali Petsiuk, Abir Das, and Kate Saenko. 2018. RISE: Randomized Input Sampling for Explanation of Black-box Models. arXiv:1806.07421 http://arxiv.org/abs/1806.07421

[27] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. 2017. Large-scale evolution of image classifiers. https://arxiv.org/abs/1703.01041

[28] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2020. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. arXiv:2006.02903 https://arxiv.org/abs/2006.02903

[29] Mélanie Rey and Andriy Mnih. 2021. Gaussian dropout as an information bottleneck layer.

[30] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1135–1144. https://doi.org/10.1145/2939672.2939778

[31] Rei Sato, Jun Sakuma, and Youhei Akimoto. 2021. AdvantageNAS: Efficient Neural Architecture Search with Credit Assignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 9489–9496.

[32] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. 2019. Evaluating the Search Phase of Neural Architecture Search. arXiv:1902.08142 http://arxiv.org/abs/1902.08142

[33] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In *Proceedings of the 23rd International Conference on World Wide Web* (Seoul, Korea) *(WWW '14 Companion)*. Association for Computing Machinery, New York, NY, USA, 373–374. https://doi.org/10.1145/2567948.2577348

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

[35] Farhana Sultana, Abu Sufian, and Paramartha Dutta. 2019. Advancements in Image Classification using Convolutional Neural Network. arXiv:1905.03288 http://arxiv.org/abs/1905.03288

[36] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[37] Randy L. Teach and Edward H. Shortliffe. 1981. An analysis of physician attitudes regarding computer-based clinical consultation systems. *Computers and Biomedical Research* 14, 6 (1981), 542–558. https://doi.org/10.1016/0010-4809(81)90012-4

[38] T. Terasvirta and H. M. Anderson. 1992. Characterizing nonlinearities in business cycles using smooth transition autoregressive models. *Journal of Applied Econometrics* 7, S1 (Dec. 1992), S119–S136. https://doi.org/10.1002/jae.3950070509

[39] Chih-Fong Tsai. 2008. Financial decision support using neural networks and support vector machines. *Expert Systems* 25, 4 (2008), 380–393. https://doi.org/10.1111/j.1468-0394.2008.00449.x

[40] Steven Walczak. 1994. Categorizing university student applicants with neural networks. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, Vol. 6. IEEE, 3680–3685.

[41] Xiaoxing Wang, Wenxuan Guo, Junchi Yan, Jianlin Su, and Xiaokang Yang. 2021. ZARTS: On Zero-order Optimization for Neural Architecture Search. arXiv:2110.04743 https://arxiv.org/abs/2110.04743

[42] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. , 279–292 pages.

[43] Bichen Wu, Forrest N. Iandola, Peter H. Jin, and Kurt Keutzer. 2017. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 446–454. https://doi.org/10.1109/CVPRW.2017.60

[44] Rui Yan, Yiping Song, and Hua Wu. 2016. Learning to Respond with Deep Neural Networks for Retrieval-Based Human-Computer Conversation System. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, Raffaele Perego, Fabrizio Sebastiani, Javed A. Aslam, Ian Ruthven, and Justin Zobel (Eds.). ACM, 55–64. https://doi.org/10.1145/2911451.2911542

[45] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. https://arxiv.org/abs/1611.01578