Designing neural network architectures using reinforcement learning

Bowen Baker, Otkrist Gupta, Nikhil Naik, Ramesh Raskar

Paper presentation by: Christopher Beckham
(COMP767)

Introduction

- Designing convolutional neural network (CNN) architectures requires human expertise and is laborious
- Design space is huge
    - # of layers, ordering of layers, receptive field size, # feature maps, strides, nonlinearity, etc…
- Can we use RL as a ‘meta-algorithm’ to select good CNN architectures?

Overall technique

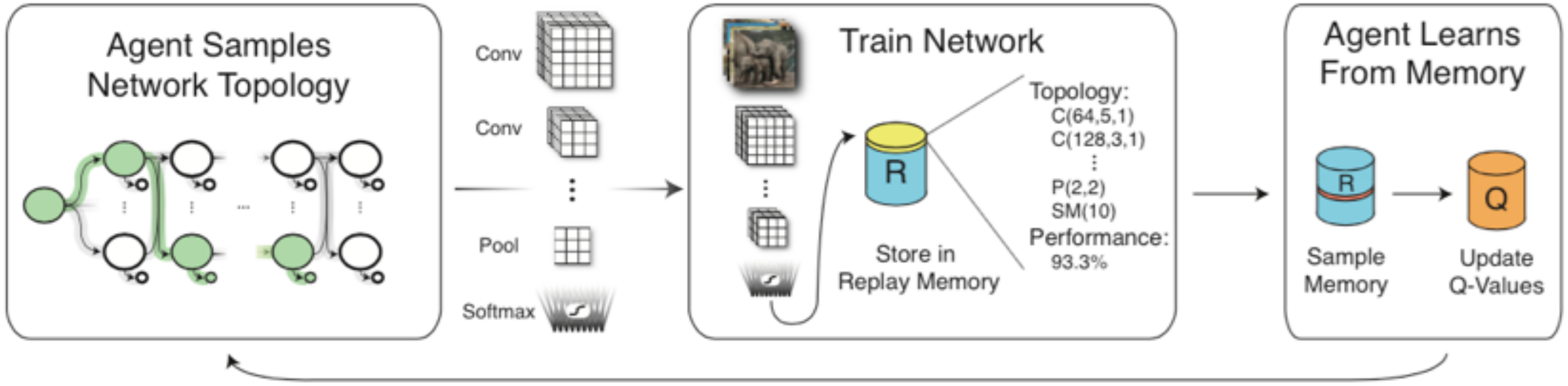
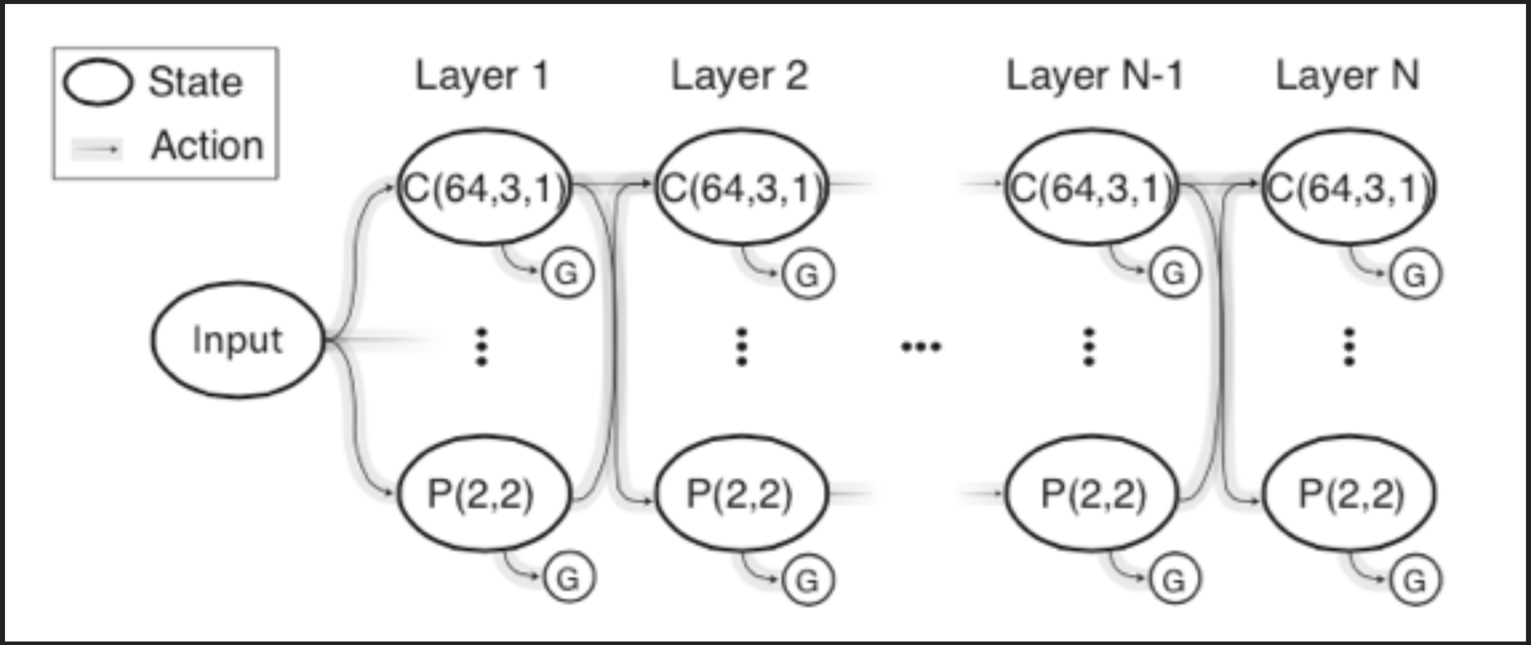


Figure 1: **Designing CNN Architectures with Q-learning:** The agent begins by sampling a Convolutional Neural Network (CNN) topology conditioned on a predefined behavior distribution and the agent's prior experience (left block). That CNN topology is then trained on a specific task; the topology description and performance, e.g. validation accuracy, are then stored in the agent's memory (middle block). Finally, the agent uses its memories to learn about the space of CNN topologies through Q-learning (right block).

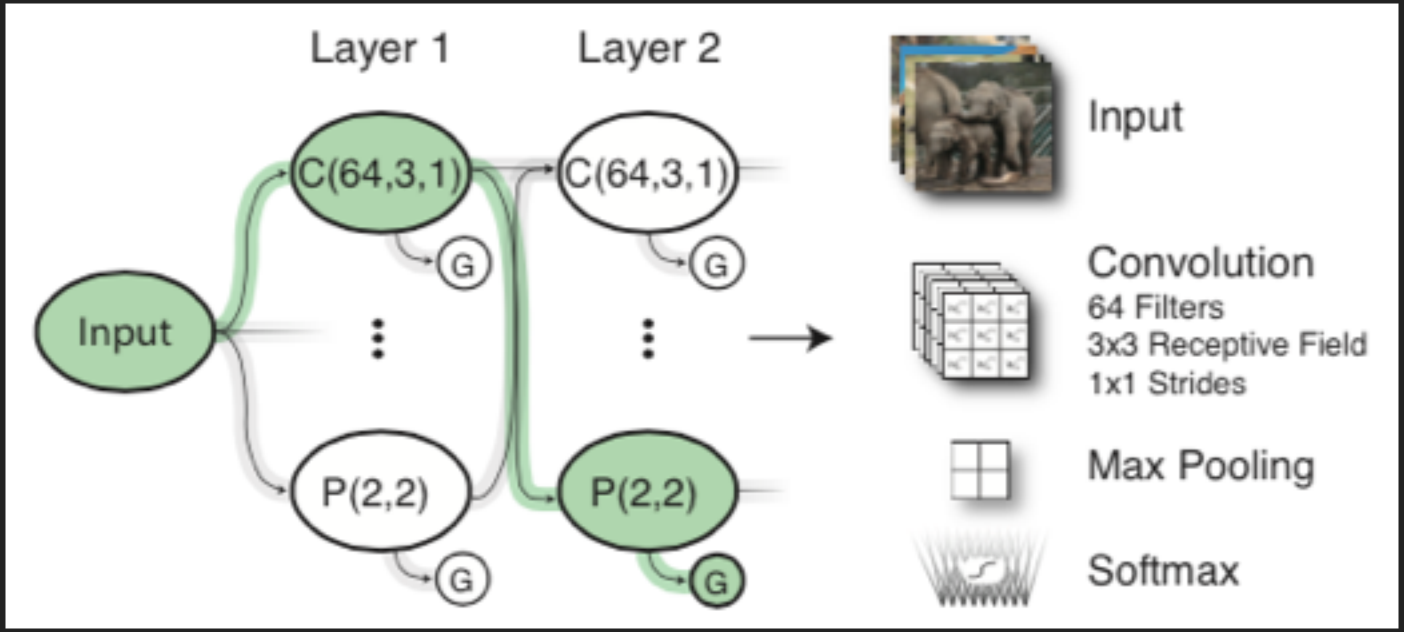
Some notation and refresher

- We have a finite state space S and action space U
- For any state $s_i \in S$, there are a finite set of actions $U(s_i) \subseteq U$
- Suppose we are in state s_i, take some action $u \in U(s_i)$, receive some reward r_t, and arrive in state s_j
- $Q_{t+1}(s_i,u) = Q_t(s_i,u) + \alpha[\ r_t + \gamma \max_{u'} Q_t(s_j,u)\ ]$
- Q-learning is off-policy
  - We can learn the optimal policy while using a completely different policy for exploration

entire state and action space:



a random trajectory:



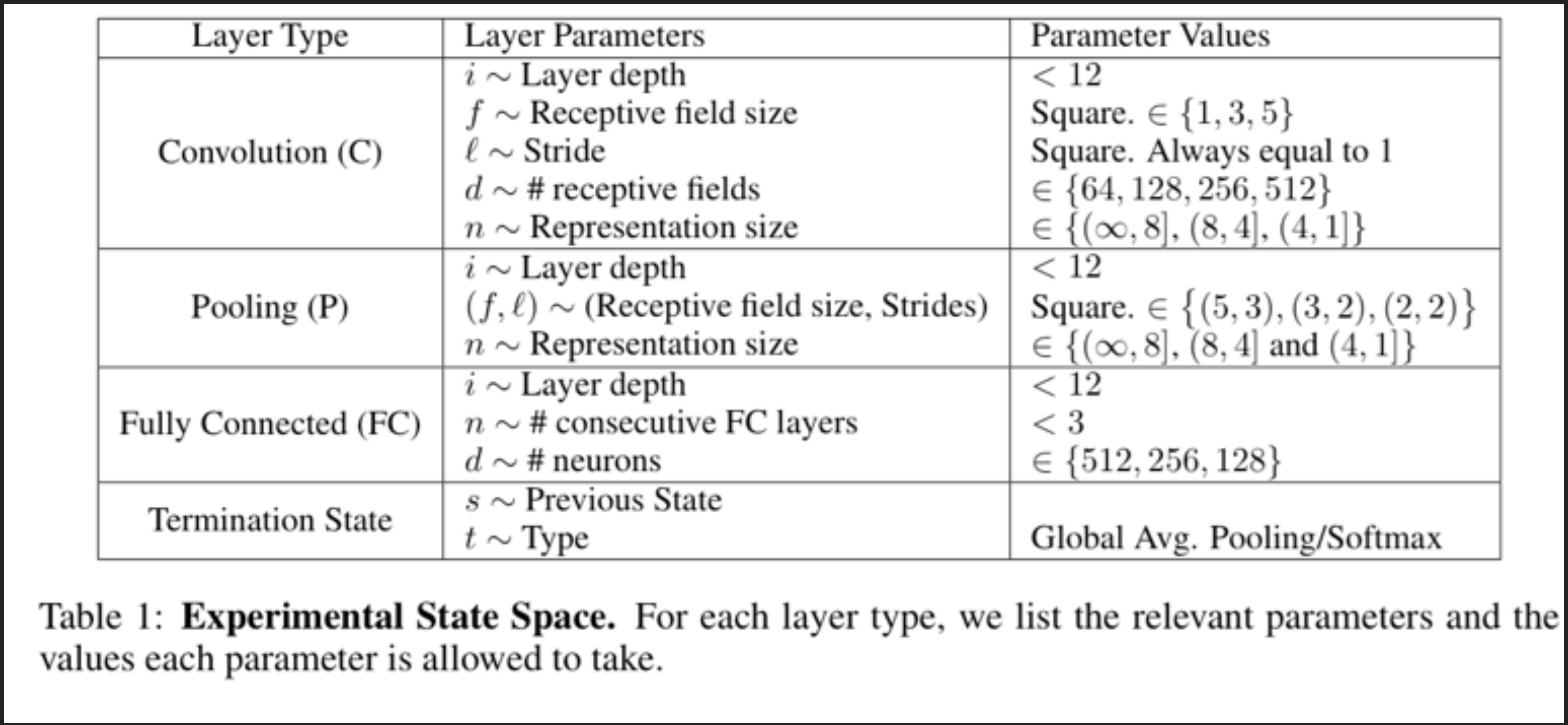
The state space

- Each state is a specification of relevant layer parameters

- Many constraints are made on actions between states

    - Mitigate state-space blow-up

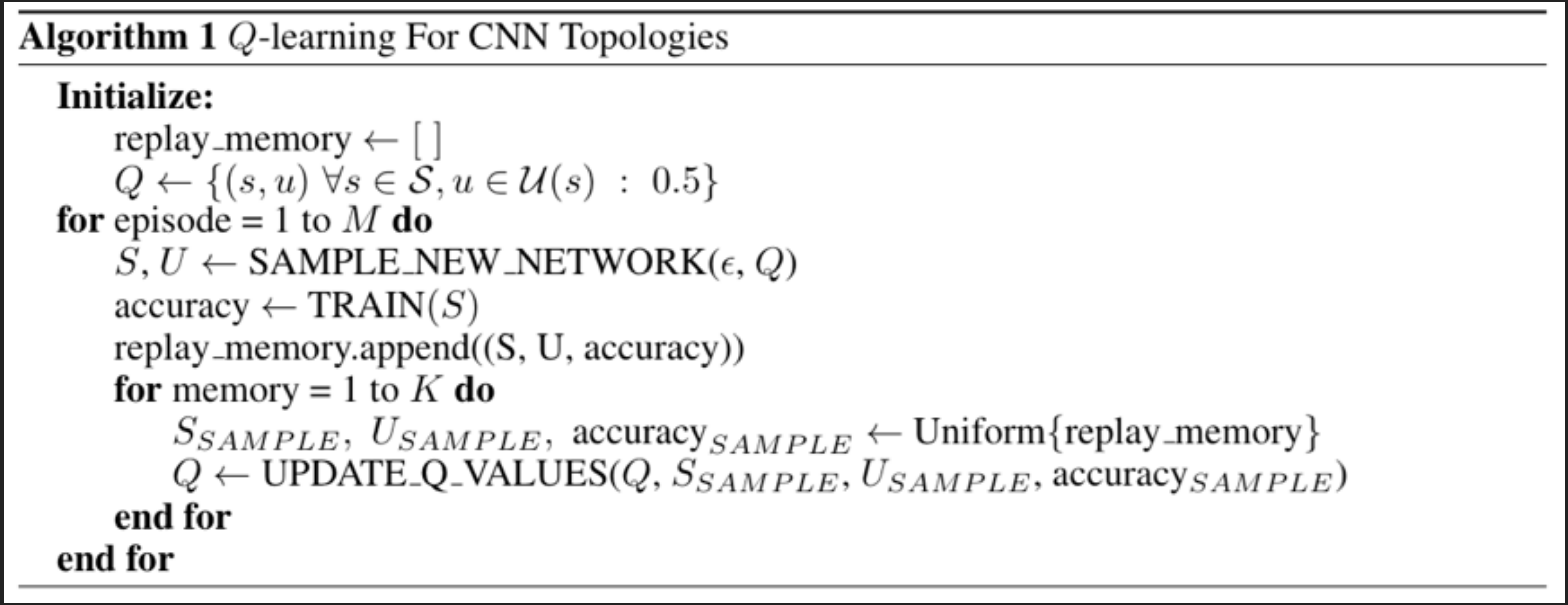
    - Ensure experiments are tractable



Layer Type	Layer Parameters	Parameter Values
Convolution (C)	$i \sim$ Layer depth $f \sim$ Receptive field size $\ell \sim$ Stride $d \sim$ # receptive fields $n \sim$ Representation size	< 12 Square. $\in \{1, 3, 5\}$ Square. Always equal to 1 $\in \{64, 128, 256, 512\}$ $\in \{(\infty, 8], (8, 4], (4, 1]\}$
Pooling (P)	$i \sim$ Layer depth $(f, \ell) \sim$ (Receptive field size, Strides) $n \sim$ Representation size	< 12 Square. $\in \{(5, 3), (3, 2), (2, 2)\}$ $\in \{(\infty, 8], (8, 4]$ and $(4, 1]\}$
Fully Connected (FC)	$i \sim$ Layer depth $n \sim$ # consecutive FC layers $d \sim$ # neurons	< 12 < 3 $\in \{512, 256, 128\}$
Termination State	$s \sim$ Previous State $t \sim$ Type	 Global Avg. Pooling/Softmax

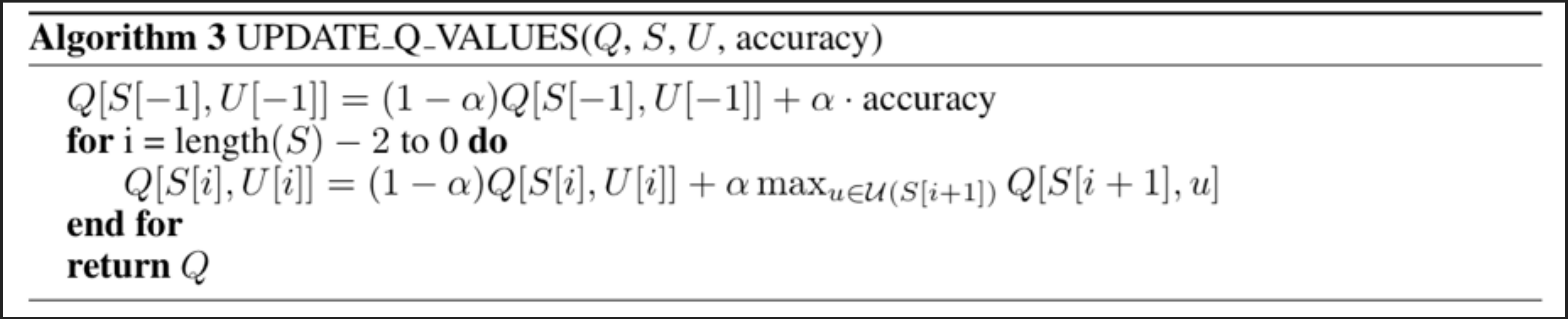
Table 1: **Experimental State Space.** For each layer type, we list the relevant parameters and the values each parameter is allowed to take.

- You can terminate from any non-termination state (terminate = add softmax layer)
- You can only transition between a state with layer depth *i* to layer depth *i+1*
- Cannot have more than two fully-connected (FC) layers
- You cannot go from a small FC layer to a bigger FC layer
- And many other constraints!



Algorithm 1 Q-learning For CNN Topologies

```
Initialize:
    replay_memory ← [ ]
    $Q \leftarrow \{(s, u)\ \forall s \in \mathcal{S}, u \in \mathcal{U}(s)\ :\ 0.5\}$
for episode = 1 to $M$ do
    $S, U \leftarrow$ SAMPLE_NEW_NETWORK($\epsilon$, $Q$)
    accuracy ← TRAIN($S$)
    replay_memory.append((S, U, accuracy))
    for memory = 1 to $K$ do
        $S_{SAMPLE}$, $U_{SAMPLE}$, accuracy$_{SAMPLE}$ ← Uniform{replay_memory}
        $Q \leftarrow$ UPDATE_Q_VALUES($Q$, $S_{SAMPLE}$, $U_{SAMPLE}$, accuracy$_{SAMPLE}$)
    end for
end for
```

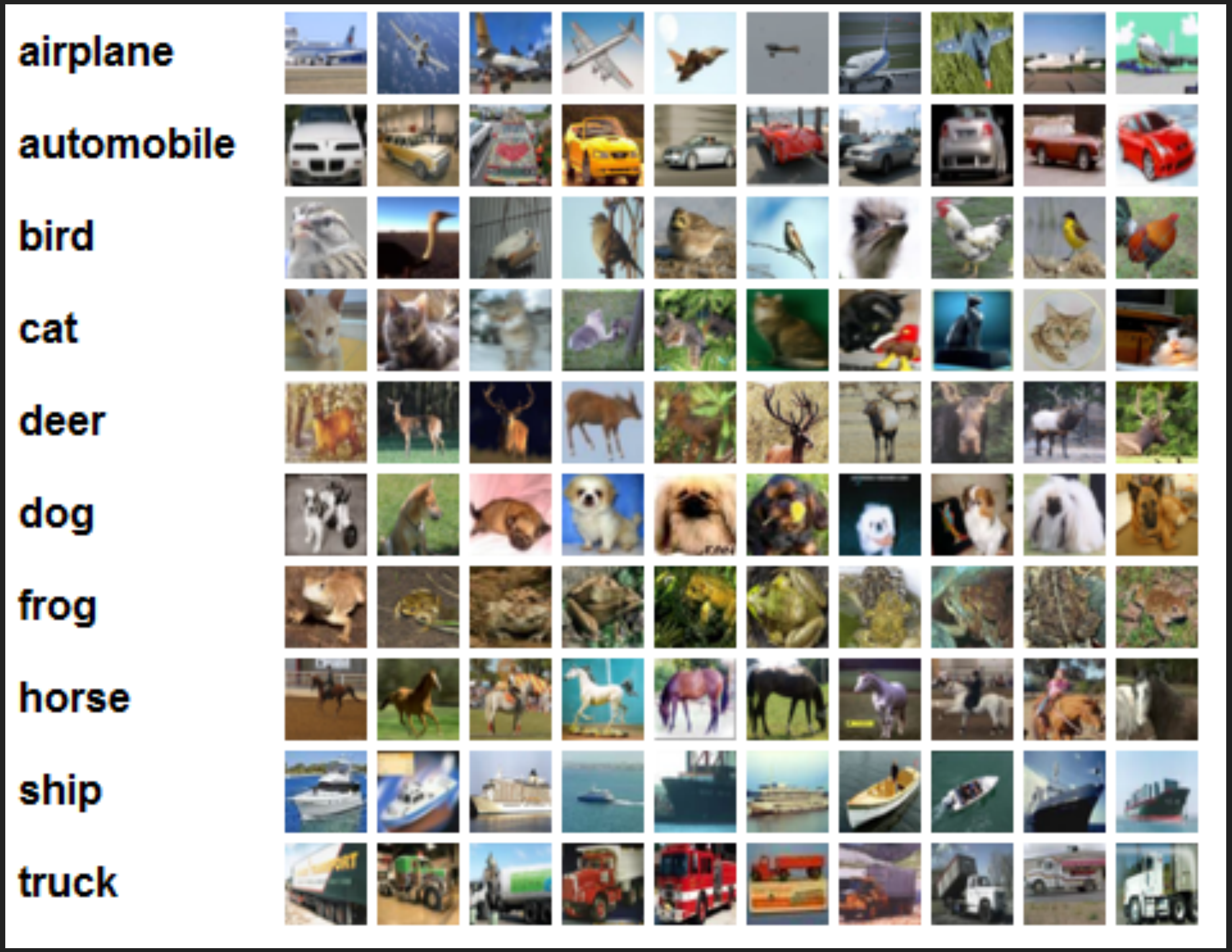


Algorithm 3 UPDATE_Q_VALUES(Q, S, U, accuracy)

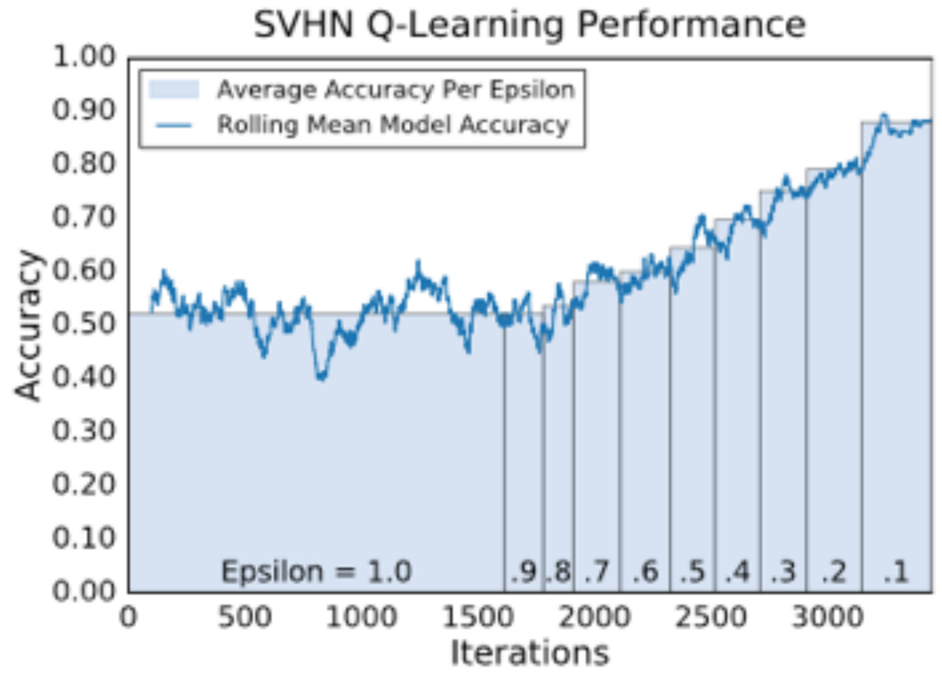
```
$Q[S[-1], U[-1]] = (1 - \alpha)Q[S[-1], U[-1]] + \alpha \cdot \text{accuracy}$
for i = length($S$) − 2 to 0 do
    $Q[S[i], U[i]] = (1 - \alpha)Q[S[i], U[i]] + \alpha \max_{u \in \mathcal{U}(S[i+1])} Q[S[i+1], u]$
end for
return $Q$
```

Datasets and experimental details

- MNIST
  - 60k in training set, 10k in testing
- Street View House Numbers (SVHN)
  - 73k in training set, 26k in test set, 531k for ‘extended’ training set
- CIFAR-10
  - 50k in training set, 10k in testing
- Two phases: ‘exploration’ and ‘fine-tuning’
  - Exploration: train quickly (e.g. train for 20 epochs using ADAM); if model doesn’t ‘learn’ within one epoch, restart training with lower initial learning rate…
  - Fine-tuning: after epsilon-schedule, select top 10 models, *and fine-tune using a longer training schedule* (this is where manual labour comes in!)



http://knowm.org/wp-content/uploads/Screen-Shot-2015-08-14-at-2.44.57-PM.png
http://ufldl.stanford.edu/housenumbers/32x32eg.png
http://karpathy.github.io/assets/cifar_preview.png



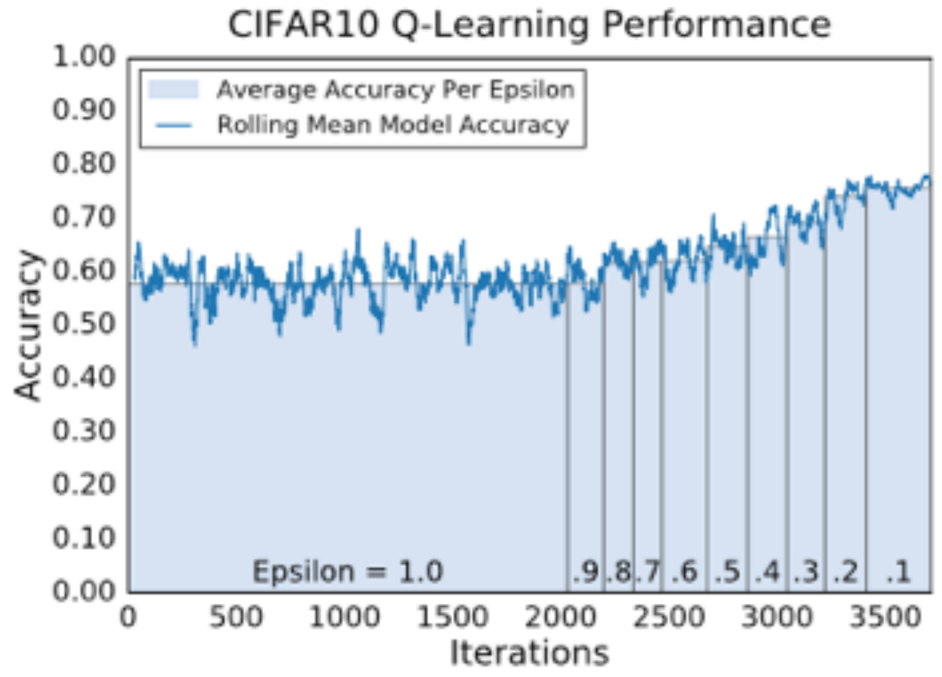
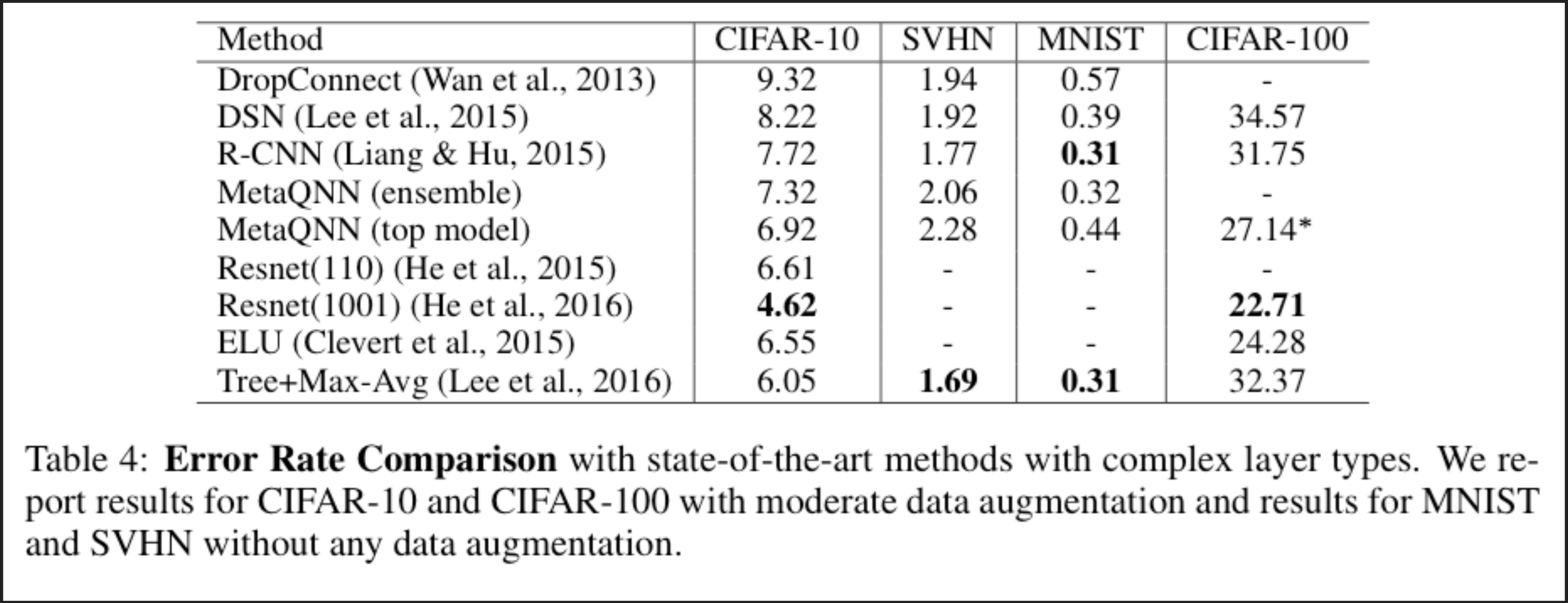


Figure 3: ***Q*-Learning Performance.** In the plots, the blue line shows a rolling mean of model accuracy versus iteration, where in each iteration of the algorithm the agent is sampling a model. Each bar (in light blue) marks the average accuracy over all models that were sampled during the exploration phase with the labeled ϵ. As ϵ decreases, the average accuracy goes up, demonstrating that the agent learns to select better-performing CNN architectures.

Method	CIFAR-10	SVHN	MNIST	CIFAR-100
Maxout (Goodfellow et al., 2013)	9.38	2.47	0.45	38.57
NIN (Lin et al., 2013)	8.81	2.35	0.47	35.68
FitNet (Romero et al., 2014)	8.39	2.42	0.51	35.04
HighWay (Srivastava et al., 2015)	7.72	-	-	-
VGGnet (Simonyan & Zisserman, 2014)	7.25	-	-	-
All-CNN (Springenberg et al., 2014)	7.25	-	-	33.71
MetaQNN (ensemble)	7.32	**2.06**	**0.32**	-
MetaQNN (top model)	**6.92**	2.28	0.44	**27.14***

Table 3: **Error Rate Comparison** with CNNs that only use convolution, pooling, and fully connected layers. We report results for CIFAR-10 and CIFAR-100 with moderate data augmentation and results for MNIST and SVHN without any data augmentation.



Method	CIFAR-10	SVHN	MNIST	CIFAR-100
DropConnect (Wan et al., 2013)	9.32	1.94	0.57	-
DSN (Lee et al., 2015)	8.22	1.92	0.39	34.57
R-CNN (Liang & Hu, 2015)	7.72	1.77	**0.31**	31.75
MetaQNN (ensemble)	7.32	2.06	0.32	-
MetaQNN (top model)	6.92	2.28	0.44	27.14*
Resnet(110) (He et al., 2015)	6.61	-	-	-
Resnet(1001) (He et al., 2016)	**4.62**	-	-	**22.71**
ELU (Clevert et al., 2015)	6.55	-	-	24.28
Tree+Max-Avg (Lee et al., 2016)	6.05	**1.69**	**0.31**	32.37

Table 4: **Error Rate Comparison** with state-of-the-art methods with complex layer types. We report results for CIFAR-10 and CIFAR-100 with moderate data augmentation and results for MNIST and SVHN without any data augmentation.

Conclusion

- Many design choices to be made in selecting a good deep network architecture
- Use reinforcement learning to select good candidate architectures for further tuning
- Competitive performance with respect to other techniques that use the same layer types
- State/action space is small and tabular Q-learning is used, but could maybe use Q-function approximation for bigger state/action spaces