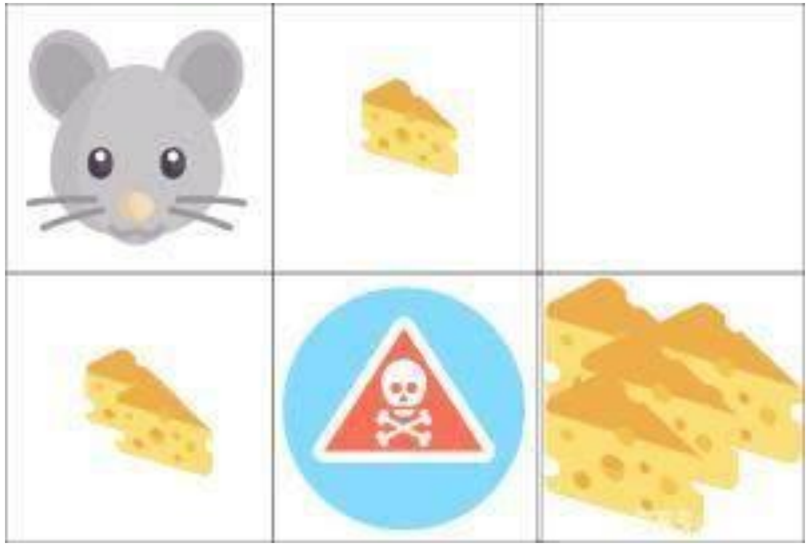
Designing Neural Network Architectures Using Reinforcement Learning

Bowen Baker, Otkrist Gupta, Nikhil Naik & Ramesh Raskar
Media Laboratory
Massachusetts Institute of Technology
Cambridge MA 02139, USA

Abstract

At present, designing convolutional neural network (CNN) architectures requires both human expertise and labor. New architectures are handcrafted by careful experimentation or modified from a handful of existing networks. We introduce MetaQNN, a meta-modeling algorithm based on reinforcement learning to automatically generate high-performing CNN architectures for a given learning task. The learning agent is trained to sequentially choose CNN layers using Q-learning with an ϵ-greedy exploration strategy and experience replay. The agent explores a large but finite space of possible architectures and iteratively discovers designs with improved performance on the learning task. On image classification benchmarks, the agent-designed networks (consisting of only standard convolution, pooling, and fully-connected layers) beat existing networks designed with the same layer types and are competitive against the state-of-the-art methods that use more complex layer types. We also outperform existing meta-modeling approaches for network design on image classification tasks.

Q-Learning



Q-Table

	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

ε-Greedy Exploration

我们指定一个探索速率ε，一开始将它设定为 1。这个就是我们将随机采用的步长。在一开始，这个速率应该处于最大值，因为我们不知道 Q-table 中任何的值。这意味着，我们需要通过随机选择动作进行大量的探索。

生成一个随机数。如果这个数大于 ε，那么我们将会进行「利用」（这意味着我们在每一步利用已经知道的信息选择动作）。否则，我们将继续进行随机探索。即ε-greedy每次以ε的概率去探索，1-ε的概率来利用。

在刚开始训练 Q 函数时，我们必须有一个大的 ε以便掌握更多的信息。随着agent对估算出的 Q 值更有把握，我们将逐渐减小 ε。

Bellman 方程

$$Q_{t+1}(s_i, u) = (1-\alpha)Q_t(s_i, u) + \alpha \left[r_t + \gamma \max\nolimits_{u' \in \mathcal{U}(s_j)} Q_t(s_j, u') \right].$$

The update equation has two parameters:
(i) α is a Q-learning rate which determines the weight given to new information over old information
(ii) γ is the discount factor which determines the weight given to short-term rewards over future rewards.

更新Q-Table

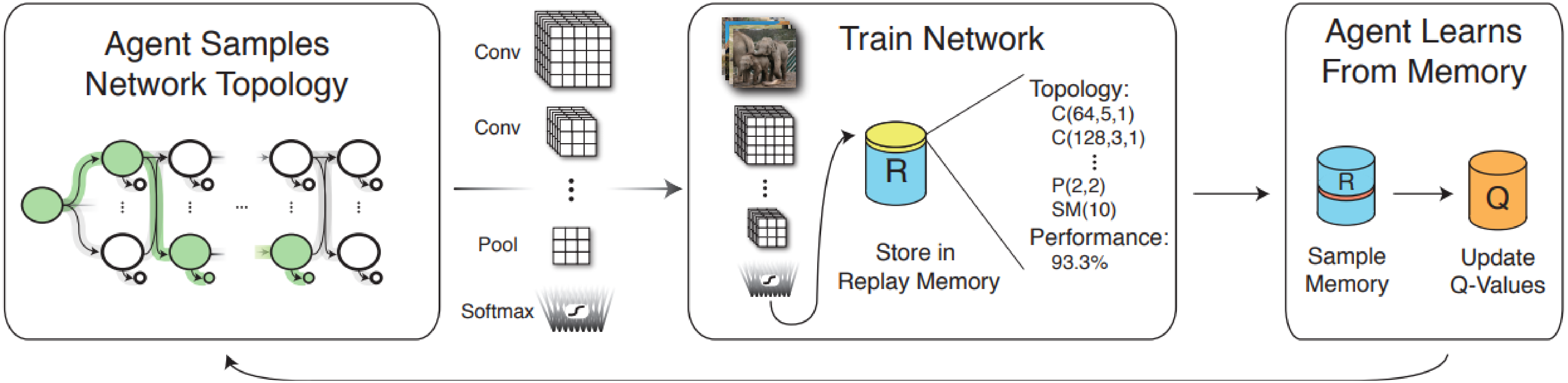
$$Q_{t+1}(s_i, u) = (1-\alpha)Q_t(s_i, u) + \alpha\left[r_t + \gamma \max_{u' \in \mathcal{U}(s_j)} Q_t(s_j, u')\right].$$

	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

	←	→	↑	↓
Start	0	0.1	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Experimental State Space

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



Algorithm 1 Q-learning For CNN Topologies

Initialize:

replay_memory $\leftarrow [\,]$

$Q \leftarrow \{(s, u)\ \forall s \in \mathcal{S}, u \in \mathcal{U}(s)\ :\ 0.5\}$

for episode = 1 to M **do**

$S, U \leftarrow$ SAMPLE_NEW_NETWORK(ϵ, Q)

accuracy $\leftarrow$ TRAIN(S)

replay_memory.append((S, U, accuracy))

for memory = 1 to K **do**

S_{SAMPLE}, U_{SAMPLE}, accuracy$_{SAMPLE} \leftarrow$ Uniform{replay_memory}

$Q \leftarrow$ UPDATE_Q_VALUES(Q, S_{SAMPLE}, U_{SAMPLE}, accuracy$_{SAMPLE}$)

end for

end for

Sample New Network

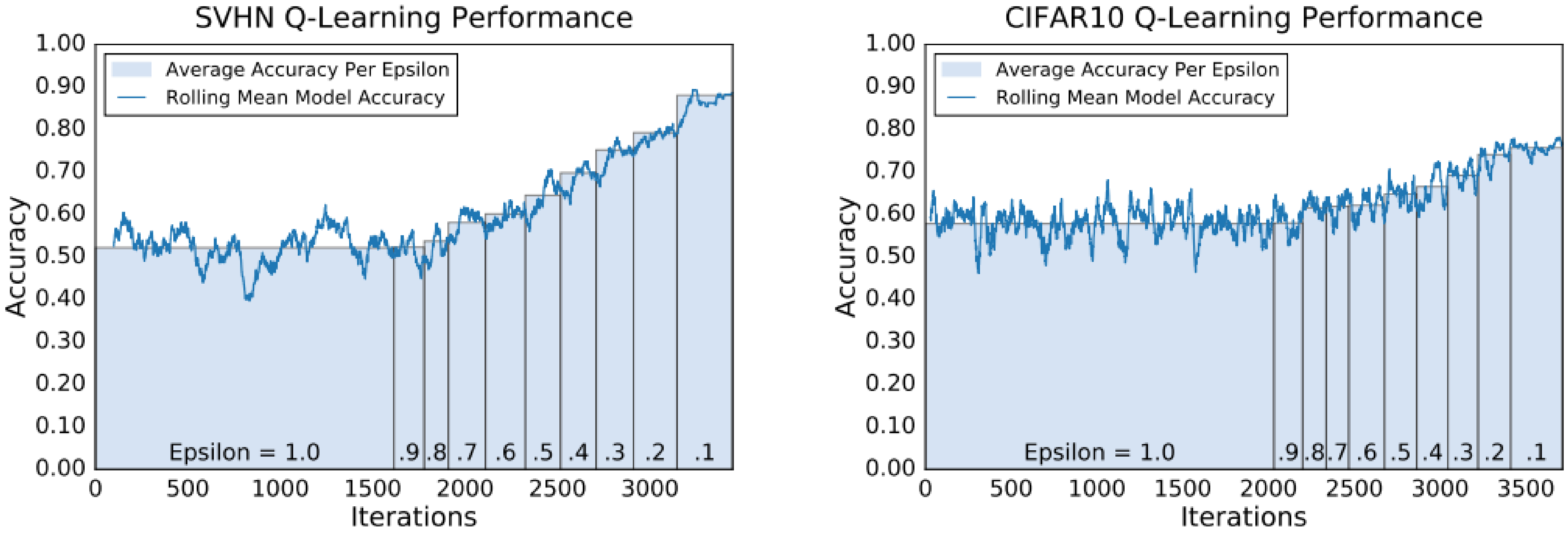
Algorithm 2 SAMPLE_NEW_NETWORK(ϵ, Q)

Initialize:
  state sequence $S = [s_{\text{START}}]$
  action sequence $U = [\ ]$
while $U[-1] \neq$ terminate **do**
  $\alpha \sim \text{Uniform}[0, 1)$
  if $\alpha > \epsilon$ **then**
    $u = \text{argmax}_{u \in \mathcal{U}(S[-1])} Q[(S[-1], u)]$
    $s' = \text{TRANSITION}(S[-1], u)$
  else
    $u \sim \text{Uniform}\{\mathcal{U}(S[-1])\}$
    $s' = \text{TRANSITION}(S[-1], u)$
  end if
  U.append(u)
  if u != terminate **then**
    S.append(s')
  end if
end while
return S, U

Update Q-Values

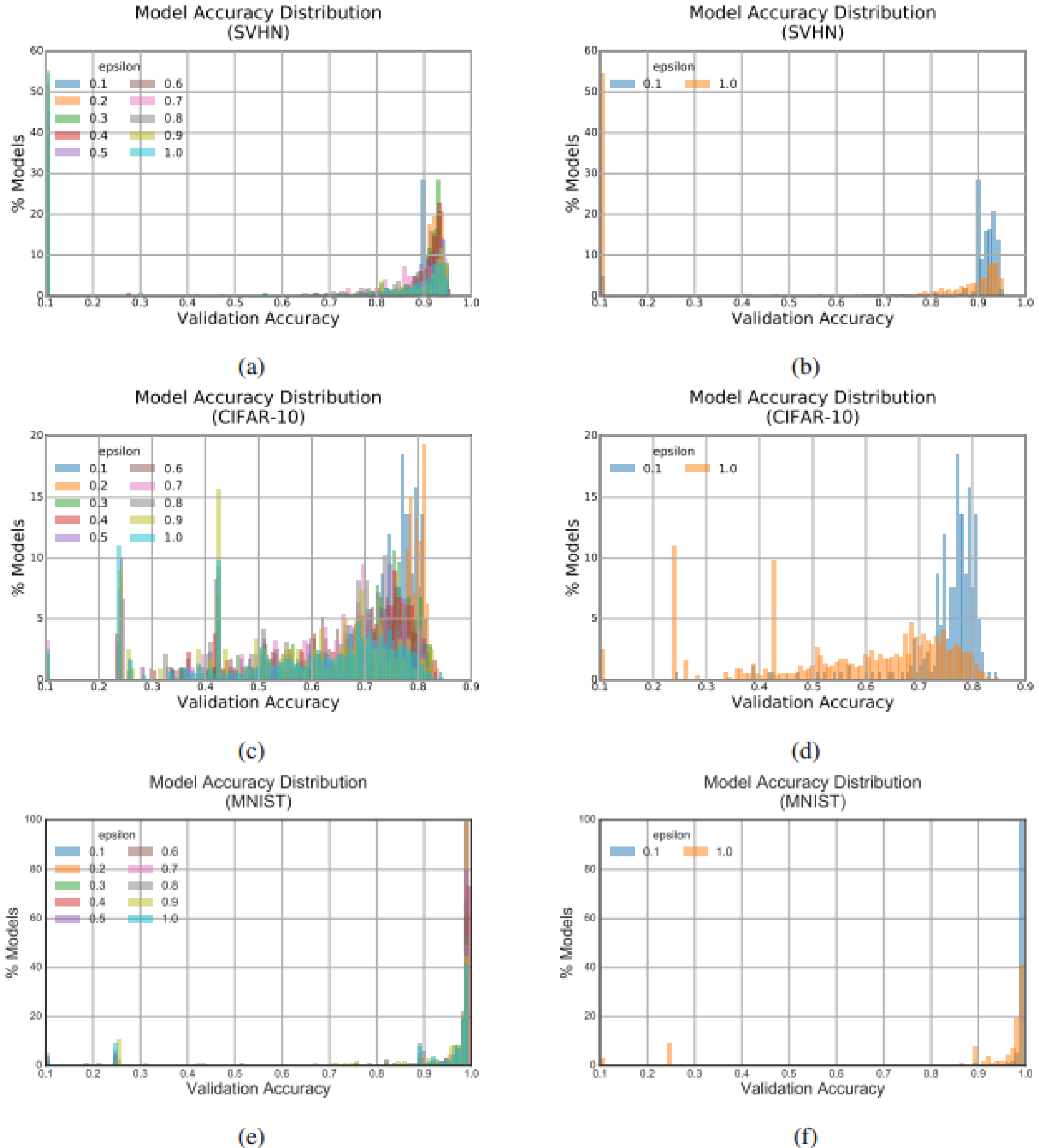
Algorithm 3 UPDATE_Q_VALUES(Q, S, U, accuracy)

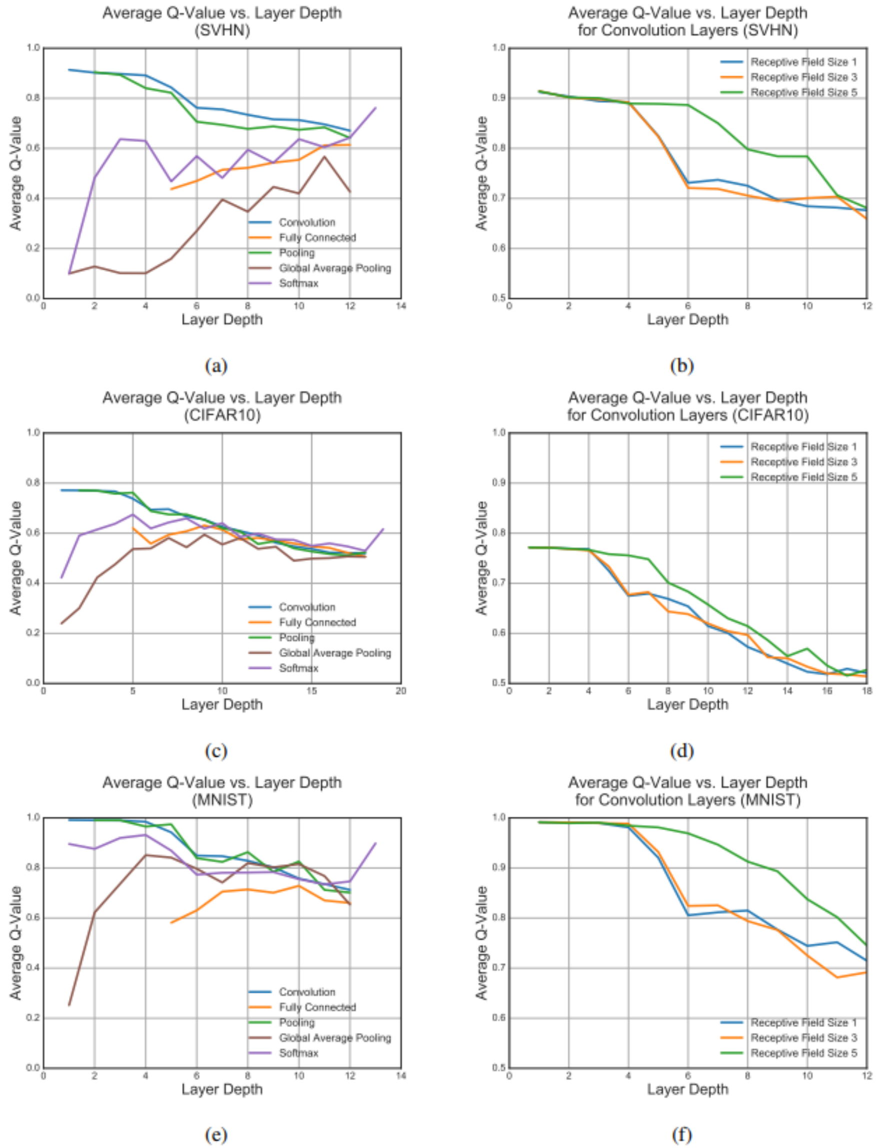
$Q[S[-1], U[-1]] = (1-\alpha)Q[S[-1], U[-1]] + \alpha \cdot \text{accuracy}$
for i = length(S) − 2 to 0 **do**
    $Q[S[i], U[i]] = (1-\alpha)Q[S[i], U[i]] + \alpha \max_{u \in \mathcal{U}(S[i+1])} Q[S[i+1], u]$
end for
return Q



ϵ	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
# Models Trained	1500	100	100	100	150	150	150	150	150	150

Accuracy Distribution versus εAnd Average Q-Value Versus Layer Depth for different layer types





Top Topologies Selected By Algorithm

Model Architecture	Test Error (%)	# Params (10^6)
[C(512,5,1), C(256,3,1), C(256,5,1), C(256,3,1), P(5,3), C(512,3,1), C(512,5,1), P(2,2), SM(10)]	6.92	11.18
[C(128,1,1), C(512,3,1), C(64,1,1), C(128,3,1), P(2,2), C(256,3,1), P(2,2), C(512,3,1), P(3,2), SM(10)]	8.78	2.17
[C(128,3,1), C(128,1,1), C(512,5,1), P(2,2), C(128,3,1), P(2,2), C(64,3,1), C(64,5,1), SM(10)]	8.88	2.42
[C(256,3,1), C(256,3,1), P(5,3), C(256,1,1), C(128,3,1), P(2,2), C(128,3,1), SM(10)]	9.24	1.10
[C(128,5,1), C(512,3,1), P(2,2), C(128,1,1), C(128,5,1), P(3,2), C(512,3,1), SM(10)]	11.63	1.66

Table A1: Top 5 model architectures: CIFAR-10.