Faster Convolutional Architecture Search for Semantic Segmentation

Rupesh Durgesh\textsuperscript{1,2}, Gerald Krell\textsuperscript{2}, Andrii Iegorov\textsuperscript{1}, Peter Schuberth\textsuperscript{1}, Felix Friedmann\textsuperscript{1}

\textsuperscript{1}Audi Electronics Venture GmbH, Germany
\{rupesh.durgesh, felix.friedmann, peter.schuberth\}@audi.de
\textsuperscript{2}Otto-von-Guericke-Universität Magdeburg, Germany
\{gerald.krell\}@ovgu.de
The reason why autonomous driving didn’t arrive (yet)

Real traffic is too complex to be solved by manually designed models

>> Machine Learning is needed to deal with this degree of complexity
Architectures for AI-powered driving

End2End Approach

Sensory Input → DNN → Steering Wheel Angle

Modular Approach

Objects → Depth → Fusion → Interpretation → Prediction → Trajectory Planning → Control → Compare

Safety Path – collision avoidance only – no AI – ASIL D
Designing DNNs is tedious...

... and requires expert knowledge:
layers, layer parameters, network topology and
Hyperparameters must be chosen

Can’t this be automated?
## State of the Art

<table>
<thead>
<tr>
<th>Method</th>
<th>Principle</th>
<th>Design space exploration</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recurrent Network Controller*</td>
<td>Recurrent NN predicts layers</td>
<td>Vast space (800 GPUs used)</td>
<td>Classification</td>
</tr>
<tr>
<td>MetaQNN*</td>
<td>Q-learning agent selects layers</td>
<td>Limited space, but slow exploration</td>
<td>Classification</td>
</tr>
<tr>
<td>Ours - FCAS</td>
<td>MetaQNN with limited design choices</td>
<td>Very limited, very greedy exploration</td>
<td>Segmentation</td>
</tr>
</tbody>
</table>

+ plenty of earlier approaches!

Semantic segmentation

Goal: determine class of every pixel in image

Segmentation CNN architecture

AlexNet for Classification

AlexNet for Segmentation

Encoder

Decoder

argmax

Deconvolution

Nr of classes

Max pooling

Max pooling

Max pooling

Max pooling

Encoder

Decoder

Striped of 4

55

224

224

55

5

5

27

27

13

13

13

13

3

3

3

3

384

384

384

256

256

4096

4096

1000

motor scooter

motor scooter

go-kart

moped

Deconvolution

## Common segmentation architectures

<table>
<thead>
<tr>
<th>Fully Convolutional Network (FCN)</th>
<th>SegNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Lower level activations added to upsampling for detail reconstruction</td>
<td>• Symmetric</td>
</tr>
<tr>
<td>• Unpooling: store pooling indices for reconstruction</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DeepLab</th>
<th>Enet</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <em>Atrous spatial pyramid pooling</em> for multi-scale recognition</td>
<td>• Best-of Net</td>
</tr>
<tr>
<td>• <em>Conditional Random Fields</em> (CRF) for Refinement</td>
<td>• Few data needed for training (no pretraining)</td>
</tr>
<tr>
<td>• Atrous spatial pyramid pooling for multi-scale recognition</td>
<td>• Very small + fast</td>
</tr>
</tbody>
</table>

---

Approach

- Q-Learning based on MetaQNN
- The agent explores design space using epsilon-greedy strategy
- Validation accuracy is the reward for reinforcement learning agent
Q-learning

• Each block is a state $s$, the agent takes an action $a$, receives a reward $r$ and transits to a new state $s'$.
• An action can be chosen from options 'left', 'right', 'top', 'down'.
• $Q$-values are updated using Bellman’s equation.

$$Q(s, a) = r + \gamma \left( \max_a \left( Q(s', a') \right) \right)$$

$r$: reward; $\gamma$: discount factor
**Q-learning**

**Epsilon(ε) - greedy strategy**
- Learning follows to two motivations:
  - Exploitation: make best decision given current knowledge
  - Exploration: get further knowledge
- Exploration rate high in the beginning of learning, then drops
- **Epsilon-greedy**: choose action not according to Q-value with $P=1-\epsilon$ and random action with $P=\epsilon$
CNN design space

- The agent iteratively selects layer configurations
- After every four layers the agent must select a downsampling layer
Layer parameters

<table>
<thead>
<tr>
<th>Layers</th>
<th>Available parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>Kernel Size: {1, 3, 5}</td>
</tr>
<tr>
<td></td>
<td>Feature Maps: {16, 32, 64, 128, 144} Stride: {1}</td>
</tr>
<tr>
<td>Pooling</td>
<td>Max Pooling</td>
</tr>
<tr>
<td></td>
<td>Pool Size: {(2, 2), (3, 3)}</td>
</tr>
<tr>
<td>Dropout</td>
<td>Rate: {0.25, 0.5}</td>
</tr>
<tr>
<td>Upsampling</td>
<td>Kernel Size: {4, 4}</td>
</tr>
<tr>
<td></td>
<td>Stride: {2}</td>
</tr>
</tbody>
</table>

Available layer configurations

Layer configuration details
- Lower feature maps values are used to limit network parameters
- Exponential Layer Unit (ELU) is used as an activation function
- Pooling layer is used for down sampling in encoder design space
- De-convolutional layer is used as upsampling layer in decoder design space
Network topology refinement

- Network topologies are refined with bypass connections for some layers randomly
- Residual* connections helps to avoid vanishing gradient problems
- SharpMask-like* connections acts as a refinement module

Greedy CNN training process

<table>
<thead>
<tr>
<th>No of Architectures</th>
<th>Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>27</td>
</tr>
</tbody>
</table>

• Inspired by Hyperband approach
• Start with 27 architectures, train for 1 epoch
  Iteratively select top 1/3 of architectures and multiply training epochs by 3

Evaluation Metric: Intersection over Union

- Intersection over Union: ratio of prediction overlap with ground truth and union of prediction and ground truth
- Mean IoU: average IoU over all classes
Results on Audi Dataset

- Training with 640x288 image resolution experiment ran to design
- 450 architectures
- Agent-designed architectures outperforms the hand-designed architecture by ~ to 4%

Misclassification: car class (red) misclassified as truck (orange)
CamVid Results

- 500 architectures explored
- Outcome: relatively small net, that beats Enet in IoU

<table>
<thead>
<tr>
<th>Architectures</th>
<th># params</th>
<th>Mean IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>SegNet-Basic</td>
<td>1.4M</td>
<td>46.3</td>
</tr>
<tr>
<td>SegNet</td>
<td>29.4M</td>
<td>50.2</td>
</tr>
<tr>
<td>Enet</td>
<td>0.37M</td>
<td>51.3</td>
</tr>
<tr>
<td>Agent designed-1</td>
<td>2.9M</td>
<td>55.9</td>
</tr>
<tr>
<td>Agent designed-2</td>
<td>1.03M</td>
<td>53.73</td>
</tr>
<tr>
<td>Agent designed-3</td>
<td>1.02M</td>
<td>52.3</td>
</tr>
</tbody>
</table>
Shortcomings / future work

• Design space is very limited, many potential parameters (e.g. Hyperparameters, activation functions, ) omitted
• Greedy training approach discards all models that train slow
• Secondary optimization goals not addressed
• Topology refinement very limited
Conclusion

• CNN architecture design can be learned
• Even with limited resources: runtime, parameters, etc.
• Architecture learning is the new feature learning