MATH 6380p Advanced Topics in Deep Learning
Nexperia Challenge

Andrea Madotto, Genta Indra Winata,
Zhaojiang Lin, Jamin Shin

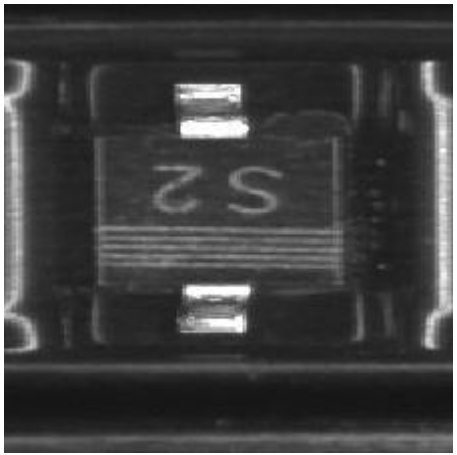
CAiRE lab

Table of Contents

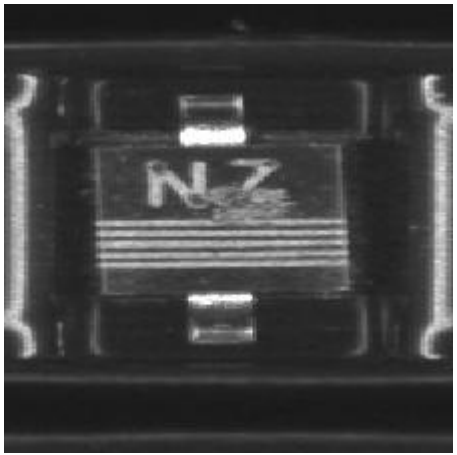
Dataset: image classification of semi-conductor

- 2 classes, **30K** training and **3K** testing images
- Validation split: 300 images (150 bad and 150 good).
- **No** particular **pre-processing** (already preprocessed)

GOOD



BAD



Baseline and results

- **Feature Based**(Res-Net feature)
  - LDA
  - Random Forest (RM)
  - Logistic Regression (LR)
  - Support Vector Machine (SVM)
- **From Scratch**
  - CNN (2d conv, 2-layer)
- **Transferring Learning**
  - Res-Net LL (last-layer)
  - Res-Net FBP (full backpropagation)

Model	Valid Acc	Test AUC
LDA	58.0	-
RF	54.0	-
LR	55.3	-
SVM	57.0	-
Res-Net LL	60.67	-
2-layer CNN	77.0	-
Res-Net FBP	**97.33**	98.30

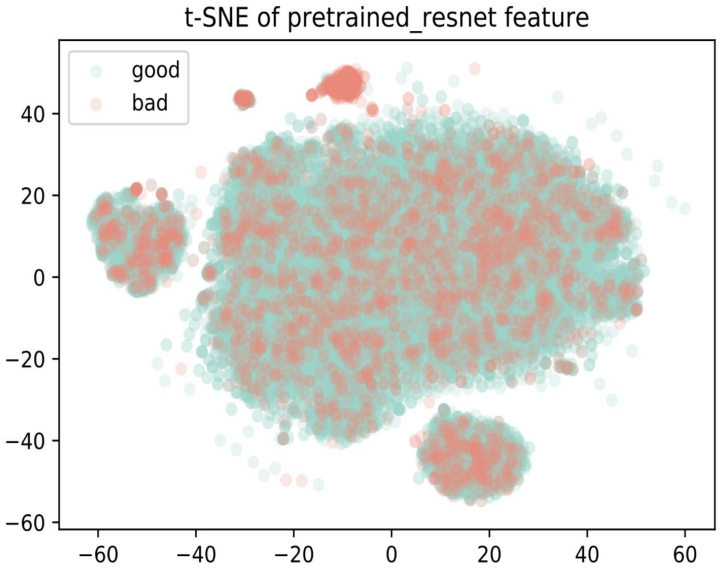
Baseline and results

- **Feature Based**(Res-Net feature)
  - LDA
  - Random Forest (RM)
  - Logistic Regression (LR)
  - Support Vector Machine (SVM)
- **From Scratch**
  - CNN (2d conv, 2-layer)
- **Transferring Learning**
  - Res-Net LL (last-layer)
  - Res-Net FBP (full backpropagation)

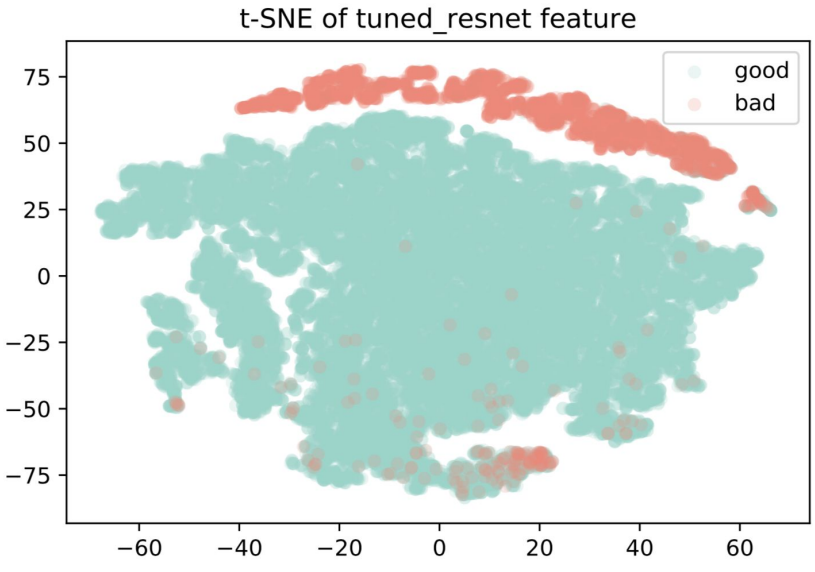
Res-Net FBP features

Model	Valid Acc	Test AUC
LDA	58.0 (96.0)	-
RF	54.0 (95.6)	-
LR	55.3 (95.6)	-
SVM	57.0 (95.6)	-
Res-Net LL	60.67	-
2-layer CNN	77.0	-
Res-Net FBP	**97.33**	98.30

T-SNE VISUALIZATION

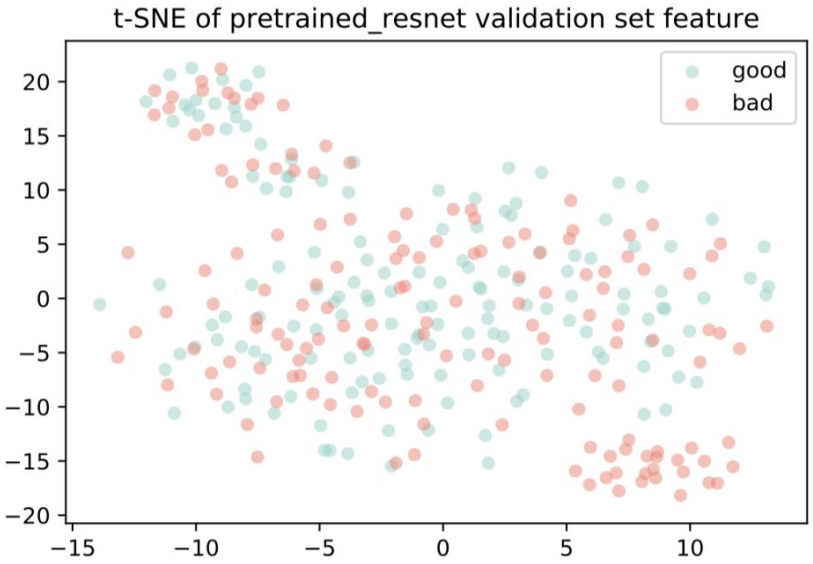


(a) Features of the pretrained Res-Net

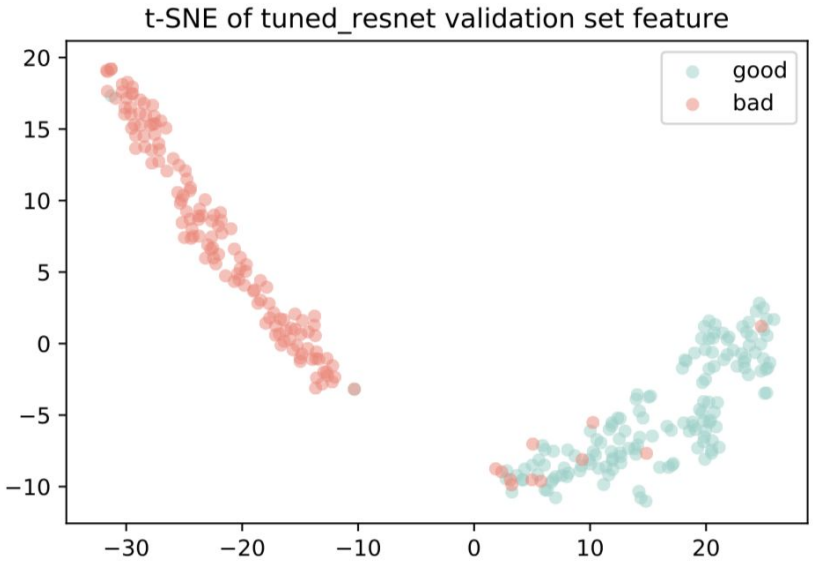


(b) Features of fine-tuned Res-Net

T-SNE VISUALIZATION



(a) Features of the pretrained Res-Net



(b) Features of fine-tuned Res-Net

Automatic architectural Search

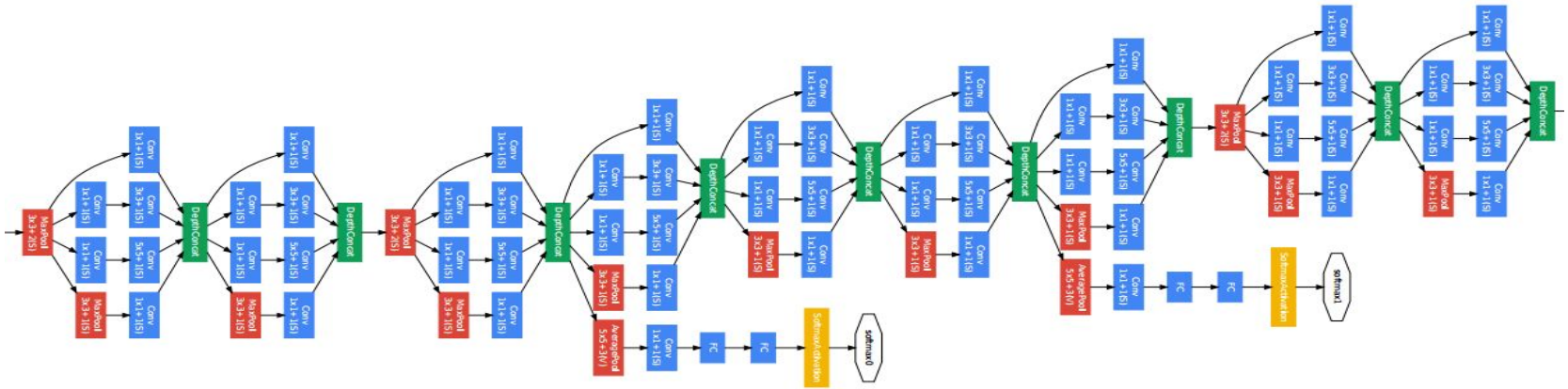
What's more?

- Transferring Learning (Res-Net FBP) is already very good
- Learning from scratch (CNN) not so good instead**
- The idea:
  - **Automatically search** an architecture that generalize well
  - Computationally expensive task:
    - Try different **hyper-parameters** (configuration) and select the one with highest validation set accuracy
    - **Optimize** (e.g. Reinforcement Learning) a function w.r.t. to configuration space
  - Several recent paper try to make it less computationally expensive:
    - Efficient Neural Architecture Search via Parameter Sharing (**ENAS**)
    - Differentiable ARchiTecture Search (**DARTS**)

** We did not tune much, but likely to overfit

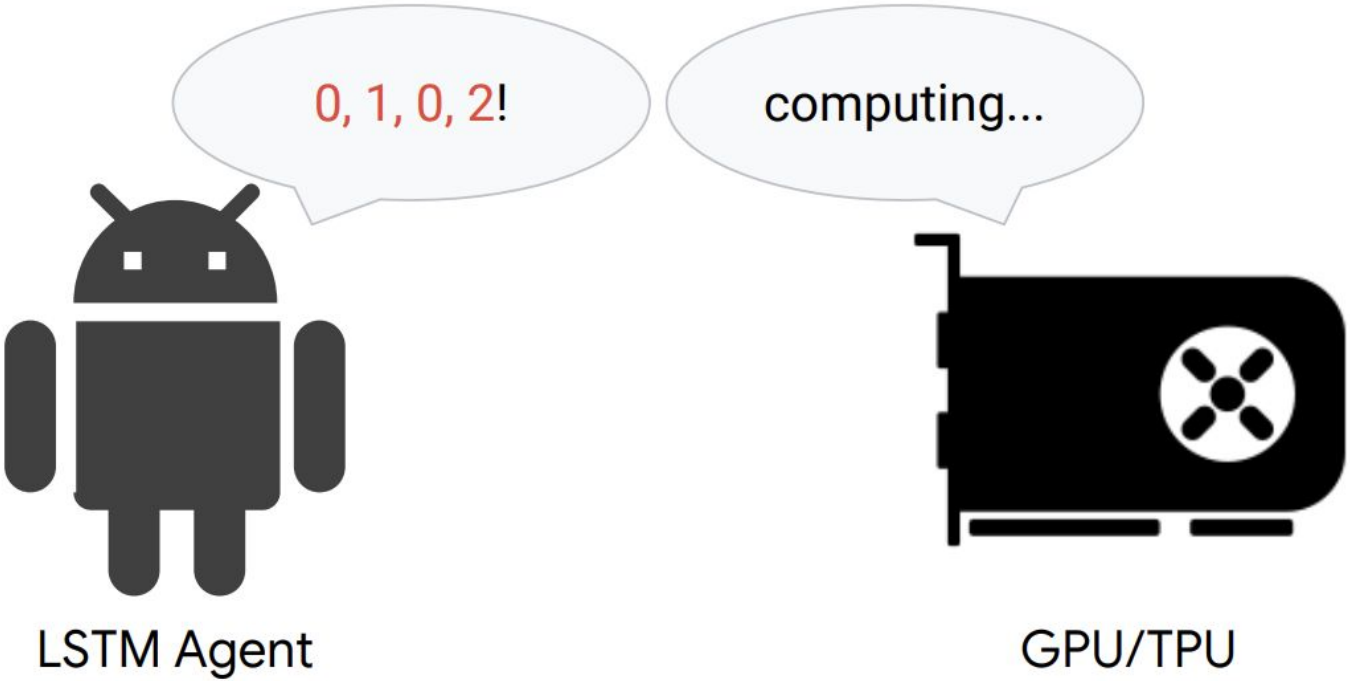
Automatic Architectural Search: What to search?

- Hyper-parameters e.g. related to learning rate, hidden size
- But for a **neural network**, many more lie in its **architecture**



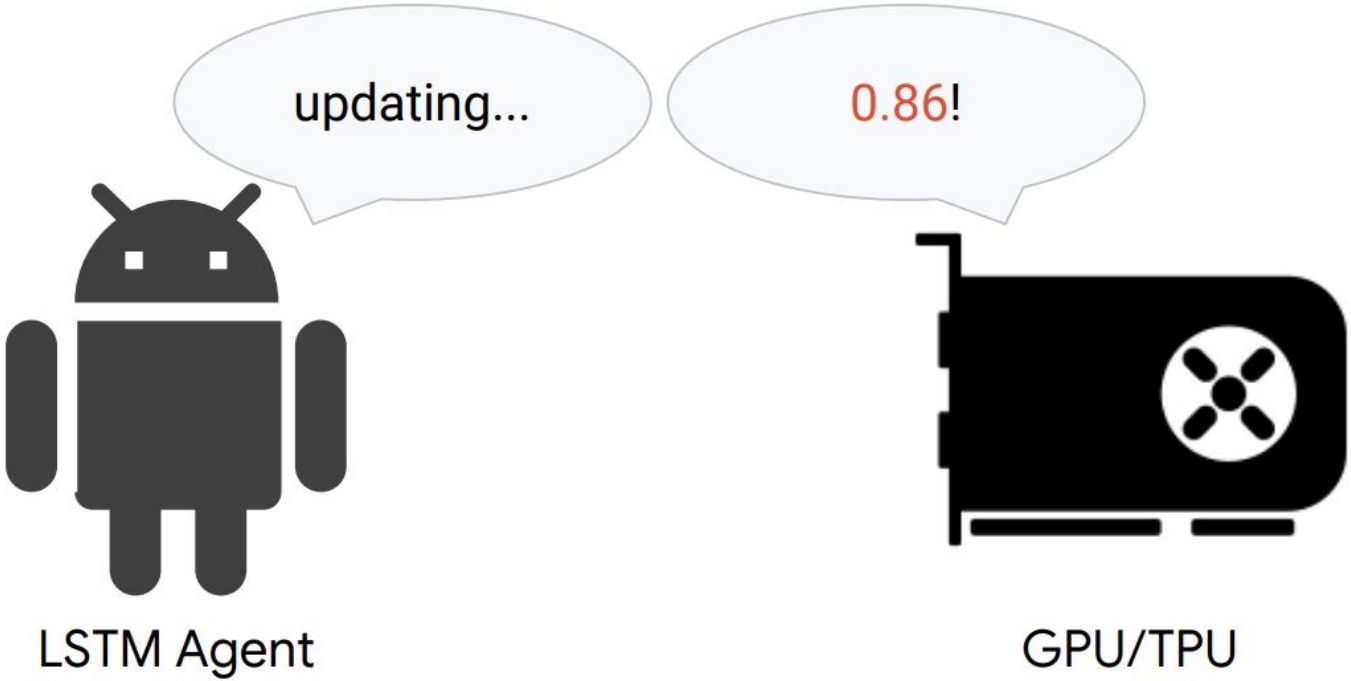
Can we **design network architectures automatically**, instead of relying on expert experience and knowledge?

Reinforcement Learning: trial and error



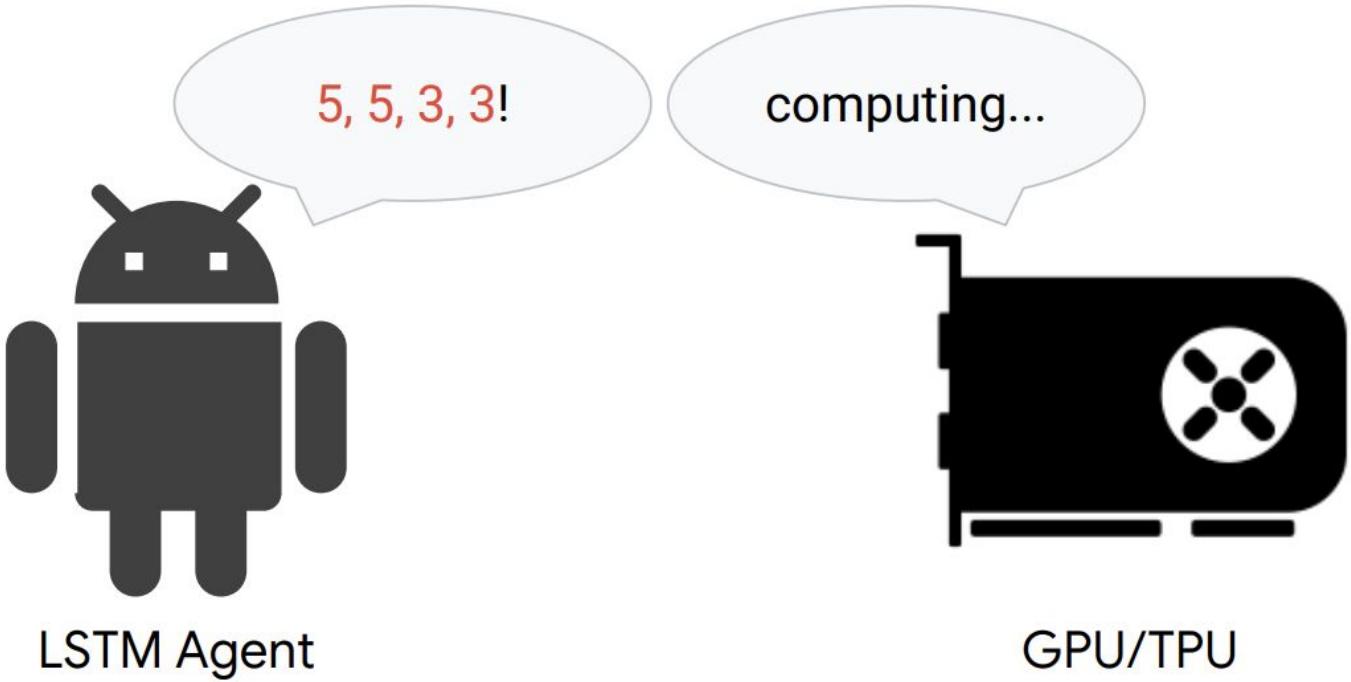
Taken from: https://cs.jhu.edu/~cxliu/slides/pnas-talk-eccv.pdf

Reinforcement Learning: trial and error



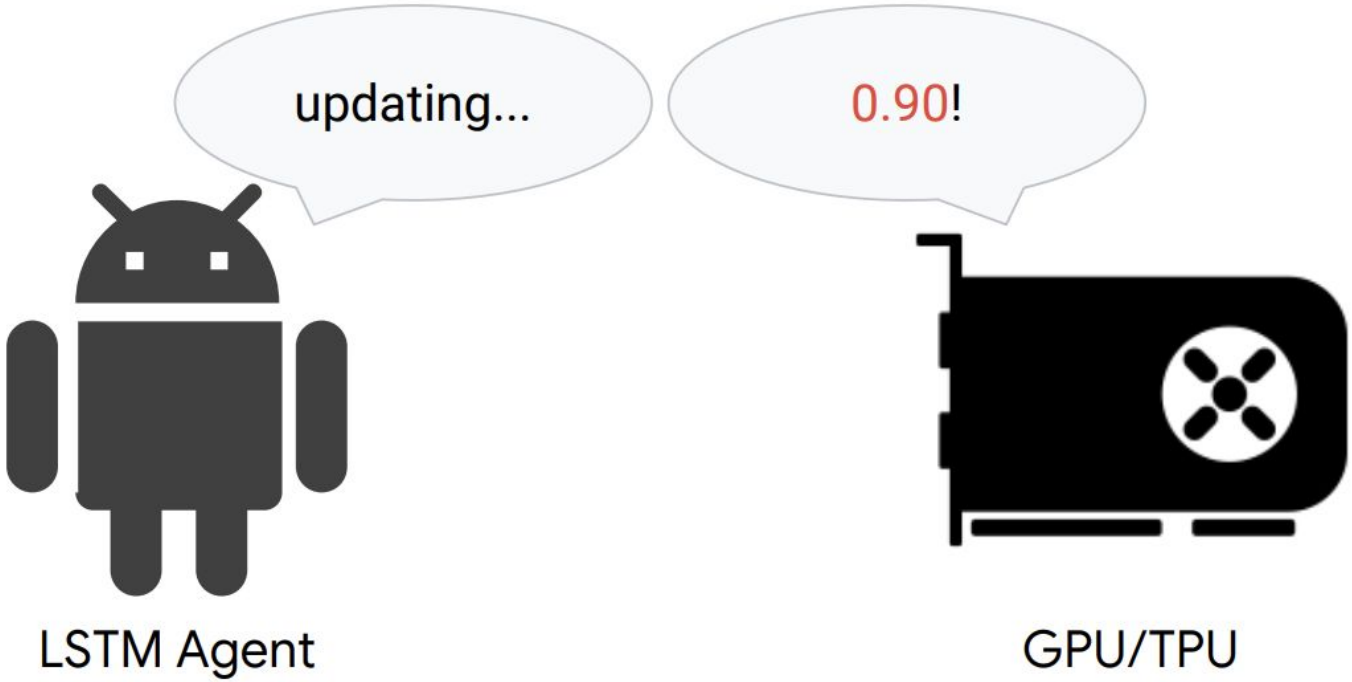
Taken from: https://cs.jhu.edu/~cxliu/slides/pnas-talk-eccv.pdf

Reinforcement Learning: trial and error



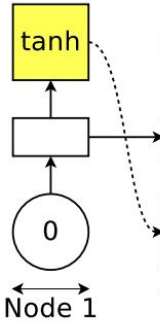
Taken from: https://cs.jhu.edu/~cxliu/slides/pnas-talk-eccv.pdf

Reinforcement Learning: trial and error

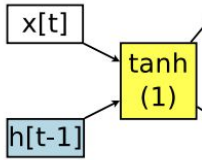


Taken from: https://cs.jhu.edu/~cxliu/slides/pnas-talk-eccv.pdf

ENAS-LSTM Agent creating RNN cell

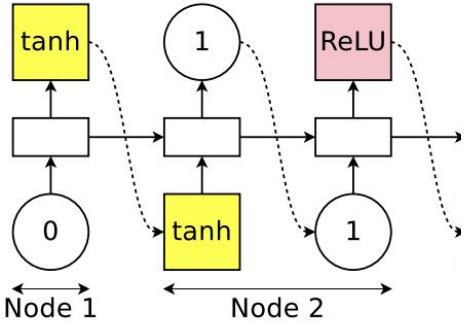


1. At node 1: The controller first samples an activation function. In our example, the controller chooses the $\tanh$ activation function, which means that node 1 of the recurrent cell should compute $h_1 = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}^{(\mathbf{x})} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(\mathbf{h})}\right)$.



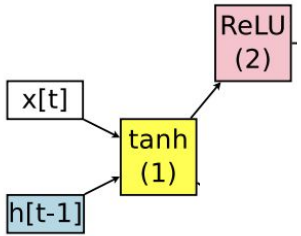
Taken from: Efficient Neural Architecture Search via Parameter Sharing (Pham et. al. 2017)

ENAS-LSTM Agent creating RNN cell



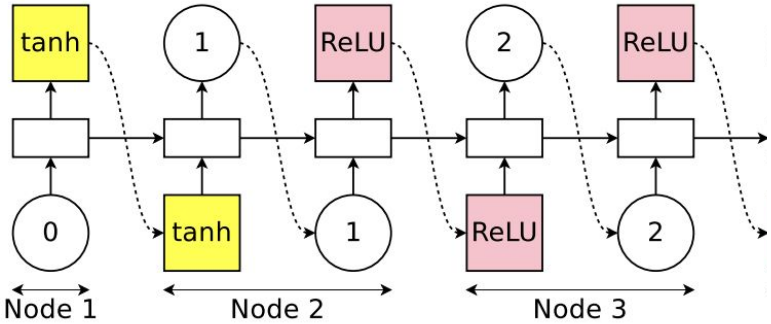
1. At node 1: The controller first samples an activation function. In our example, the controller chooses the tanh activation function, which means that node 1 of the recurrent cell should compute $h_1 = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}^{(\mathbf{x})} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(\mathbf{h})}\right)$.

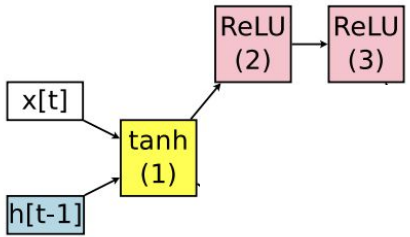
2. At node 2: The controller then samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function ReLU. Thus, node 2 of the cell computes $h_2 = \mathrm{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(\mathbf{h})})$.



Taken from: Efficient Neural Architecture Search via Parameter Sharing (Pham et. al. 2017)

ENAS-LSTM Agent creating RNN cell





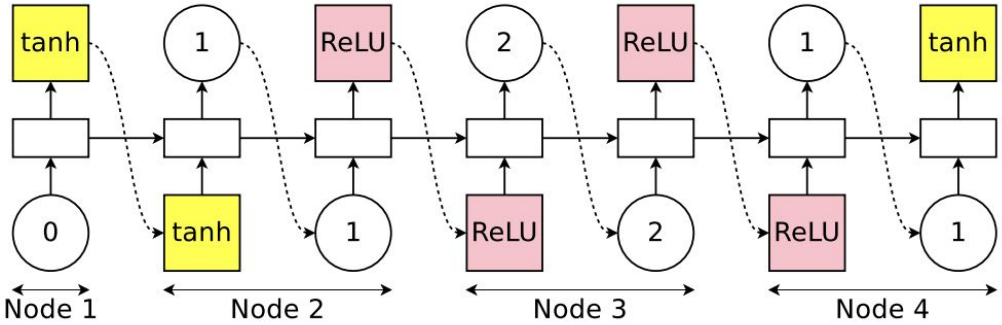
1. At node 1: The controller first samples an activation function. In our example, the controller chooses the tanh activation function, which means that node 1 of the recurrent cell should compute $h_1 = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}^{(\mathbf{x})} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(\mathbf{h})}\right)$.

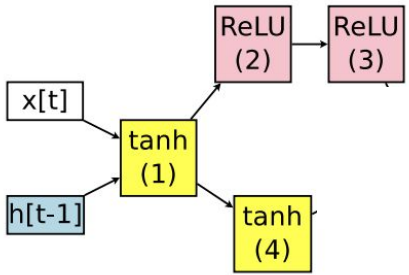
2. At node 2: The controller then samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function ReLU. Thus, node 2 of the cell computes $h_2 = \text{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(\mathbf{h})})$.

3. At node 3: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 2 and the activation function ReLU. Therefore, $h_3 = \text{ReLU}(h_2 \cdot \mathbf{W}_{3,2}^{(\mathbf{h})})$.

Taken from: Efficient Neural Architecture Search via Parameter Sharing (Pham et. al. 2017)

ENAS-LSTM Agent creating RNN cell





1. At node 1: The controller first samples an activation function. In our example, the controller chooses the tanh activation function, which means that node 1 of the recurrent cell should compute $h_1 = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}^{(\mathbf{x})} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(\mathbf{h})}\right)$.

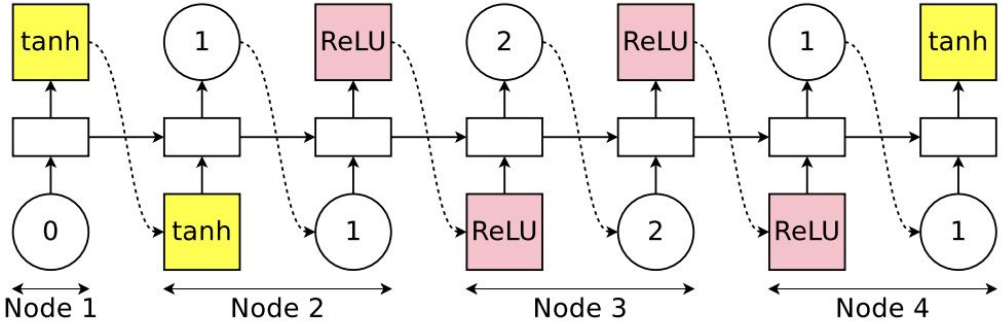
2. At node 2: The controller then samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function ReLU. Thus, node 2 of the cell computes $h_2 = \text{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(\mathbf{h})})$.

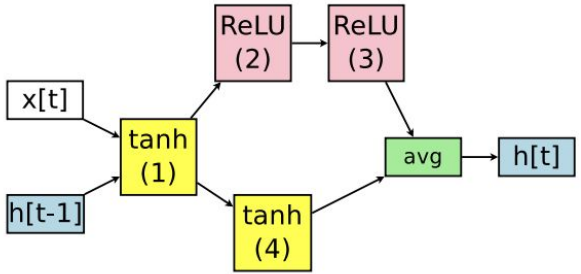
3. At node 3: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 2 and the activation function ReLU. Therefore, $h_3 = \text{ReLU}(h_2 \cdot \mathbf{W}_{3,2}^{(\mathbf{h})})$.

4. At node 4: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function tanh, leading to $h_4 = \tanh\left(h_1 \cdot \mathbf{W}_{4,1}^{(\mathbf{h})}\right)$.

Taken from: Efficient Neural Architecture Search via Parameter Sharing (Pham et. al. 2017)

ENAS-LSTM Agent creating RNN cell





1. At node 1: The controller first samples an activation function. In our example, the controller chooses the tanh activation function, which means that node 1 of the recurrent cell should compute $h_1 = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}^{(\mathbf{x})} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(\mathbf{h})}\right)$.

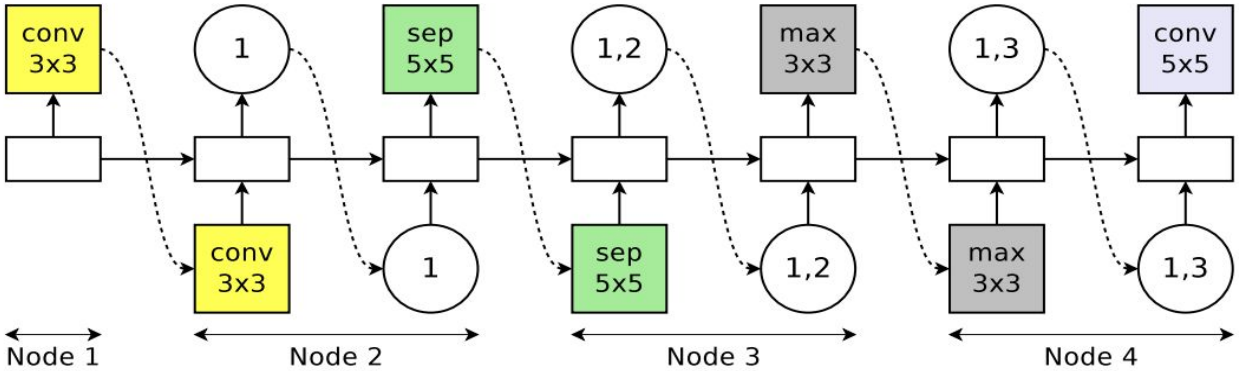
2. At node 2: The controller then samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function ReLU. Thus, node 2 of the cell computes $h_2 = \text{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(\mathbf{h})})$.

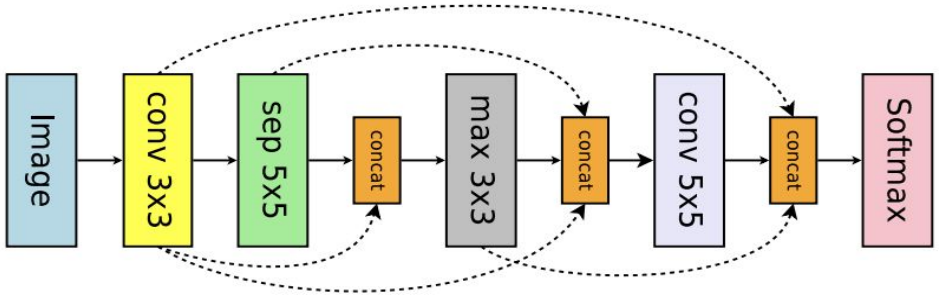
3. At node 3: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 2 and the activation function ReLU. Therefore, $h_3 = \text{ReLU}(h_2 \cdot \mathbf{W}_{3,2}^{(\mathbf{h})})$.

4. At node 4: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function tanh, leading to $h_4 = \tanh\left(h_1 \cdot \mathbf{W}_{4,1}^{(\mathbf{h})}\right)$.

5. For the output, we simply average all the loose ends, *i.e.* the nodes that are not selected as inputs to any other nodes. In our example, since the indices 3 and 4 were never sampled to be the input for any node, the recurrent cell uses their average $(h_3 + h_4)/2$ as its output. In other words, $\mathbf{h}_t = (h_3 + h_4)/2$.

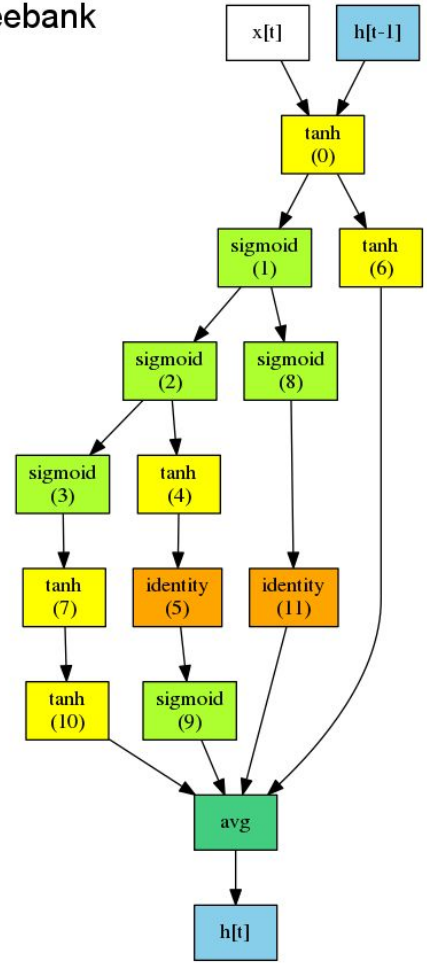
Similarly for CNNs



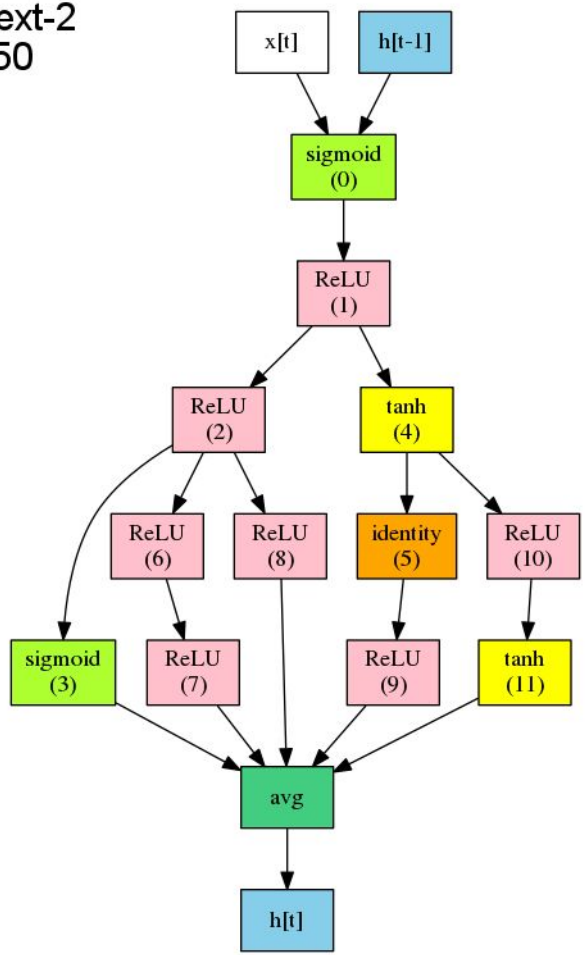


Taken from: Efficient Neural Architecture Search via Parameter Sharing (Pham et. al. 2017)

Penn Treebank
step: 50



WikiText-2
step: 50



What to optimize?

The LSTM controller act as a **policy network.**

So the optimization **alternate** the training of the architecture sampled from the policy (denoted by m) with parameter ω. Using:

$$\nabla_{\omega} \mathbb{E}_{\mathbf{m} \sim \pi(\mathbf{m};\theta)} [\mathcal{L}(\mathbf{m}; \omega)] \approx \frac{1}{M} \sum_{i=1}^{M} \nabla_{\omega} \mathcal{L}(\mathbf{m}_i, \omega)$$

And then policy optimization, using standard REINFORCE (William 1992) algorithm.

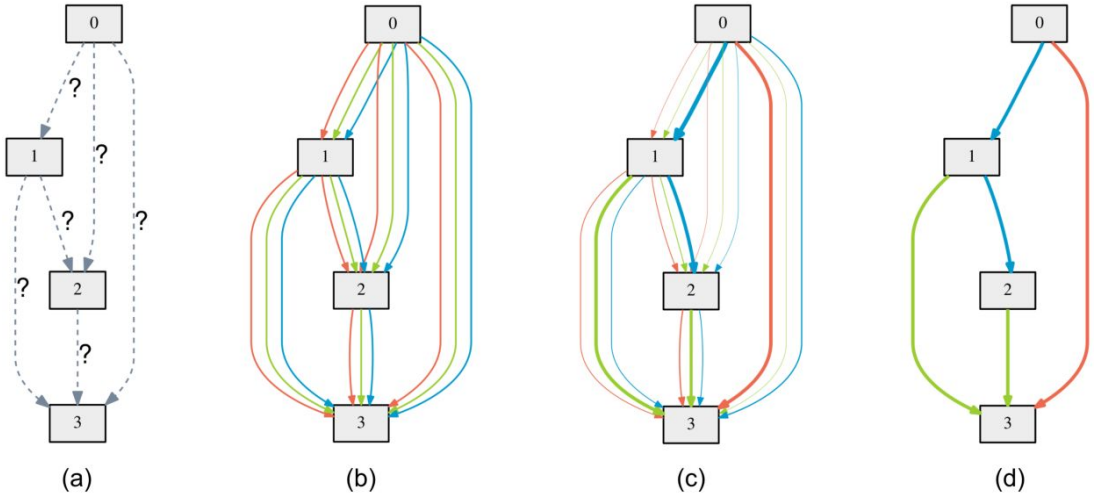
$$\mathbb{E}_{\mathbf{m} \sim \pi(\mathbf{m};\theta)} [\mathcal{R}(\mathbf{m}, \omega)]$$

Darts

Differentiable Architecture Search method (DARTS)

A gradient-based optimization method to efficiently search the best model architecture by relaxing the discrete search space to continuous space.

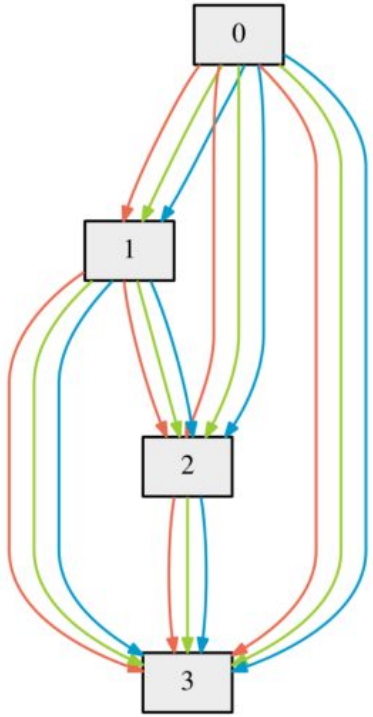
- No controllers
- No hyper-parameters
- No predictors
- Differentiable



Search Space

Lets define **a computation cell as the building block** as a directed acyclic graph.

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



Where $x^{(i)}$ is the node (e.g. a feature map in CNN)

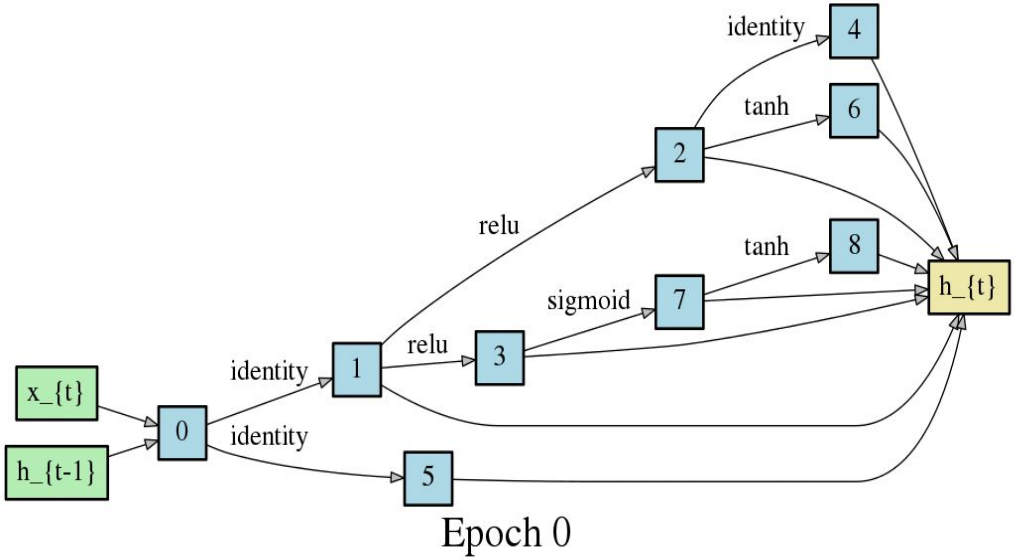
and $o^{(i,j)}$ is the operation (edge)

Within a cell

Operations

CNN:
'max_pool_3x3',
'avg_pool_3x3',
'skip_connect',
'sep_conv_3x3',
'sep_conv_5x5',
'dil_conv_3x3',
'Dil_conv_5x5'

RNN:
'none',
'tanh',
'relu',
'sigmoid',
'identity'



Continuous Relaxation and Optimization

Let $\mathcal{O}$ be a set of candidate operations. Softmax function is applied over all possible operation.

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

The operation mixing weights for a pair of nodes (i, j) are parameterized by a vector $\alpha^{(i,j)}$ of dimension $|O|$

Continuous Relaxation and Optimization

At the end of search, a discrete architecture is obtained by replacing each mixed operation

$$o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \ \alpha_o^{(i,j)}$$

Analogous to architecture search using RL or evolution where the validation set performance is treated as the reward or fitness

DARTS aims to optimize the validation loss, but using gradient descent.

Bilevel Optimization

Denote $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ the training and the validation loss. The goal for architecture search is to find α^* , minimizes the validation loss $\mathcal{L}_{val}(w^*, a^*)$

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ \text{s.t.} \quad & w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha) \end{aligned}$$

keeping the weights w_k fixed, we update the architecture so as to minimize the the validation loss after a single step of gradient descent w.r.t. the weights:

$$\mathcal{L}_{val}(w_k - \xi \nabla_w \mathcal{L}_{train}(w_k, \alpha_{k-1}), \alpha_{k-1})$$

Bilevel Optimization

In short, the architecture gradient is given by differentiating the previous eq.

$$\nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$$

Where

$$w' = w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha)$$

This approach is very similar to MAML (Model-agnostic meta-learning). With a goal to solve new learning tasks using only a small number of training samples.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400, 2017

Approximation

The gradient contains a matrix-vector product in its second term, which is expensive to compute.

$$\nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$$

The gradient can be substantially reduced from $O(|\alpha||w|)$ to $O(|\alpha| + |w|)$ using the **finite difference approximation**.

$$\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon}$$

$$w^+ = w + \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \qquad w^- = w - \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha).$$

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i,j)

while *not converged* **do**

1. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
2. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

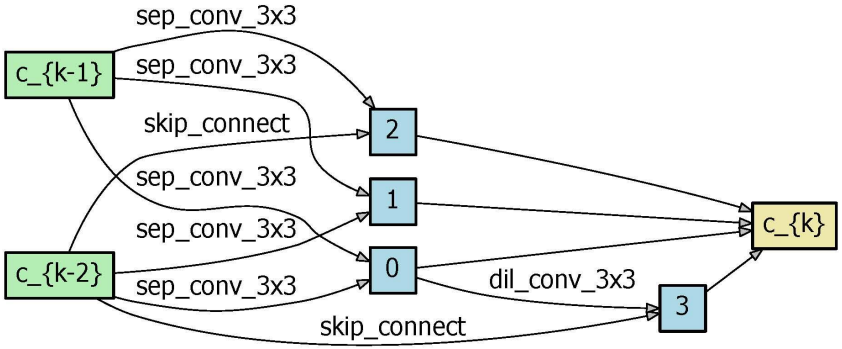
Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \, \alpha_o^{(i,j)}$ for each edge (i,j)

Architecture

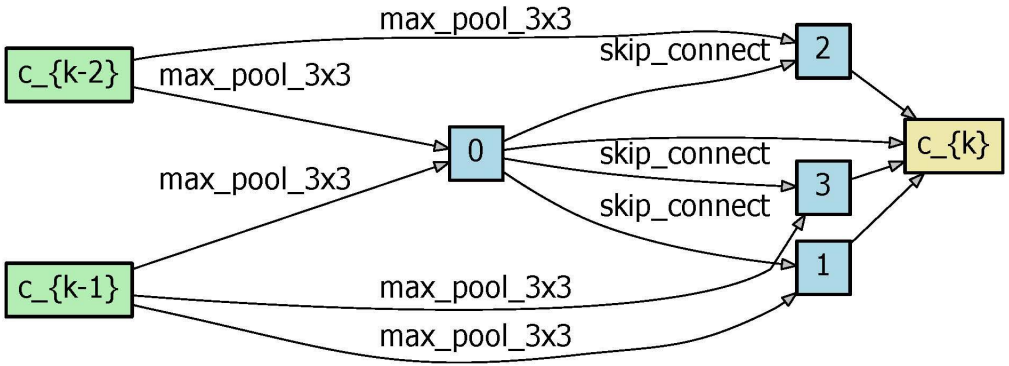
The first and second nodes of **cell k** are set to the outputs of **cell k-2** and **cell k-1**. Cells located at the 1/3 and 2/3 of the total depth of the network are reduction cells.

$(\alpha_{normal}, \alpha_{reduction})$ are shared in normal and reduction cells.

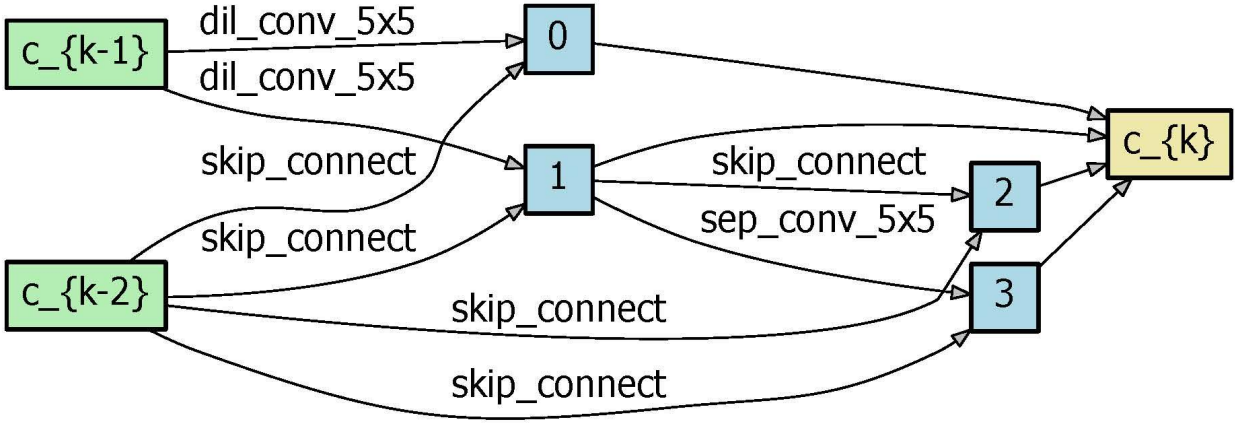
Normal cell on ImageNet



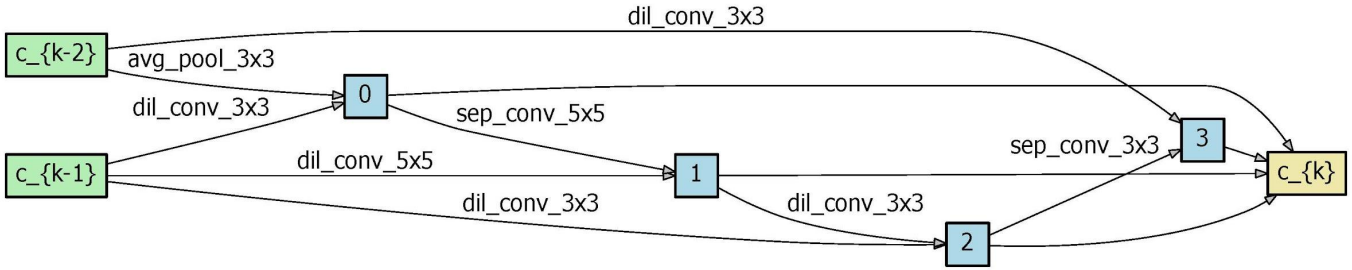
Reduction cell on ImageNet



Normal cell on Nexperia



Reduction cell on Nexperia



Darts results

Model	Valid Acc	Test AUC
LDA	58.0 (96.0)	-
RF	54.0 (95.6)	-
LR	55.3 (95.6)	-
SVM	57.0 (95.6)	-
Res-Net LL	60.67	-
2-layer CNN	77.0	-
Res-Net FBP	**97.33**	98.30
DARTS ImageNet	95.33	**99.06**
DARTS Nexperia	95.67	98.57

CAM-GradCAM

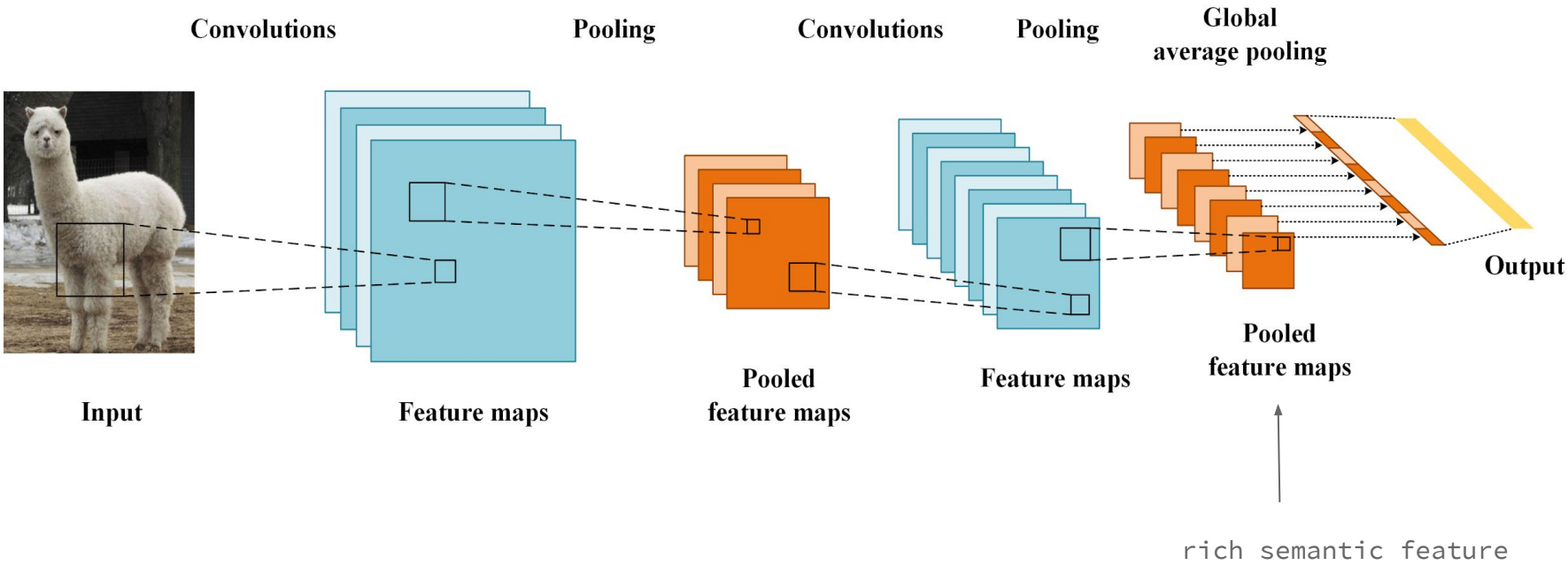
Visualization (CAM vs grad-CAM)

How to interpret the result we get?

How to make sure CNN learn to extract correct features?

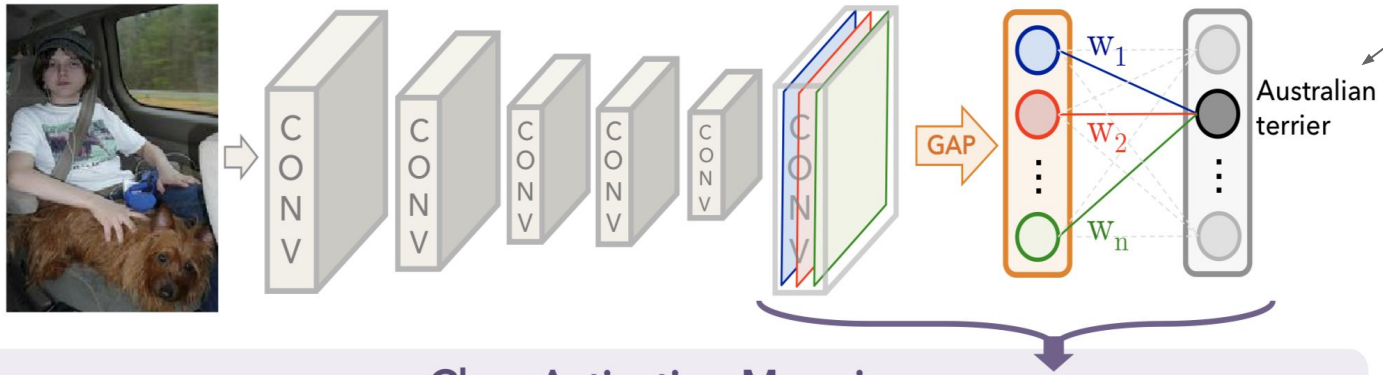
How to visualize the “attention” of CNN?

Visualization (CAM vs grad-CAM)

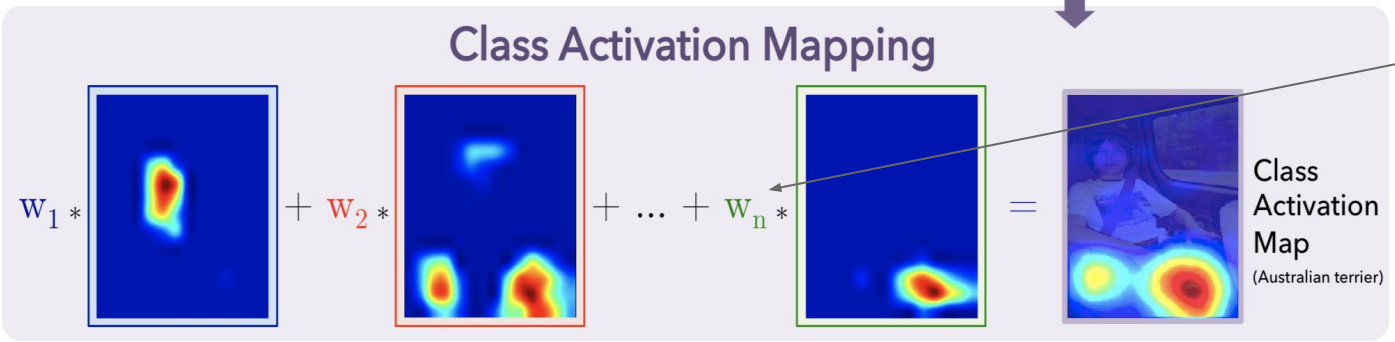


Taken from: https://bindog.github.io/blog/2018/02/10/model-explanation

Class activation capping -CAM



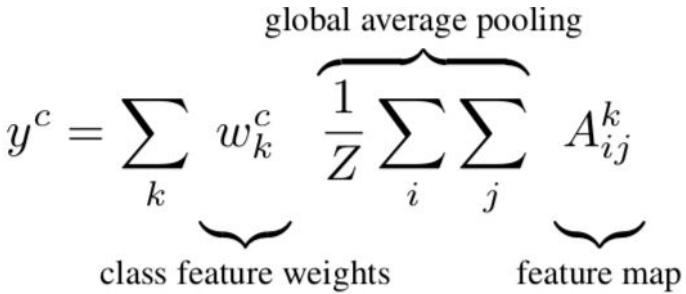
Why Australian terrier?



W tell you which feature is more important

Taken from: B.Zhou et al Learning Deep Features for Discriminative Localization. *CVPR*, 2016.

Class activation capping -CAM

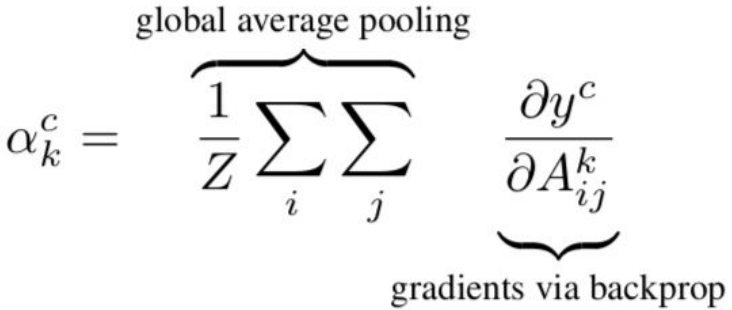


$$L_{\text{CAM}}^c = \underbrace{\sum_k w_k^c A^k}_{\text{linear combination}} .$$

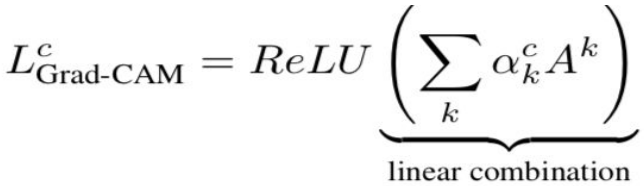
Shortcoming: need to be re-trained, hard to visualize when retraining is very expensive

Taken from: Grad-cam: why did you say that? visual explanations from deep networks via gradient-based localization.

Gradient-weighted Class activation capping (grad-CAM)

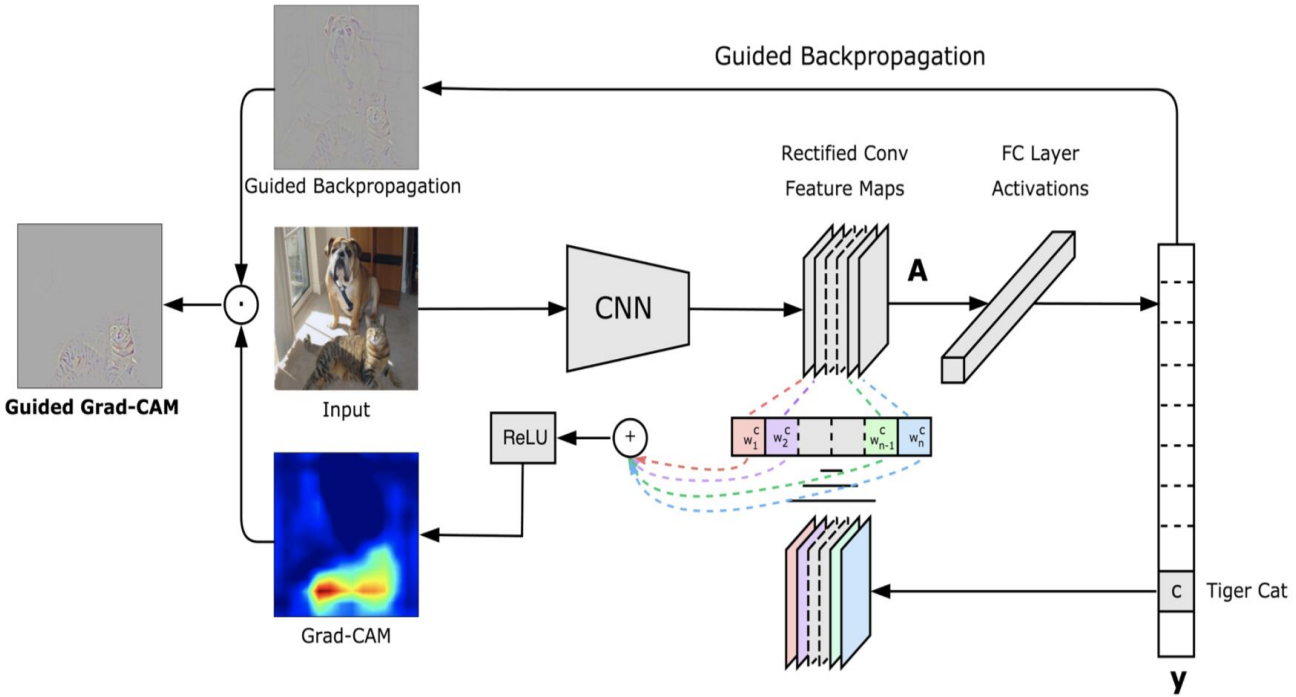


Instead of obtain feature map weights from retrained linear layer, Grad-CAM **compute** the **gradient** of the **y** with respect to feature maps **A**.



Use **ReLU** to **normalize** the weighted combination of feature maps. The motivation for the **ReLU** is the following: we are only interested in the features that have a ***positive*** **influence** on the class of interest

Gradient-weighted Class activation capping (grad-CAM)



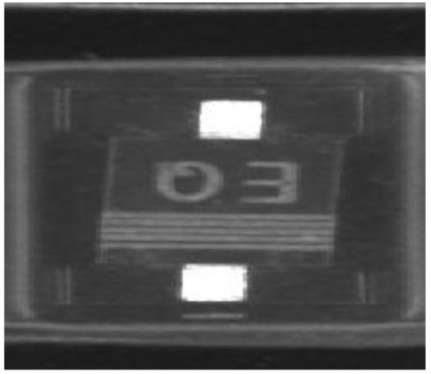


Figure 6: Sample labeled as good in the training set

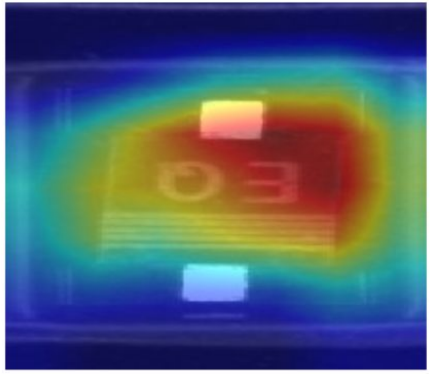


Figure 7: After backpropagation using correct labels

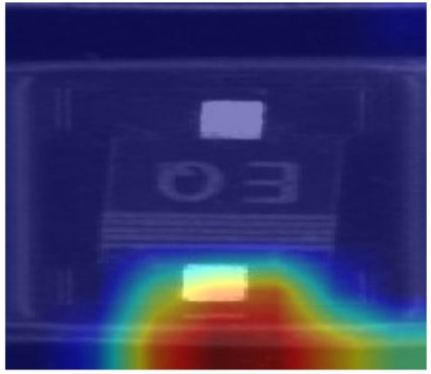


Figure 8: After backpropagation using wrong labels

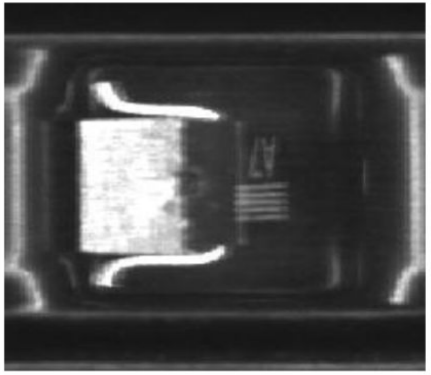


Figure 9: Sample labeled as bad in the training set

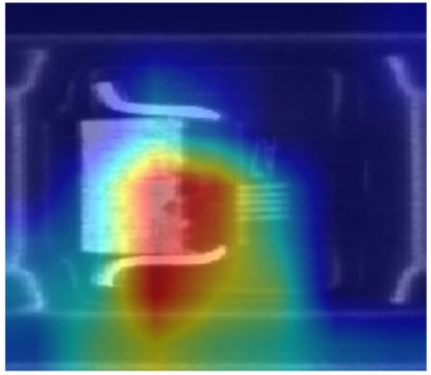


Figure 10: After backpropagation using correct labels

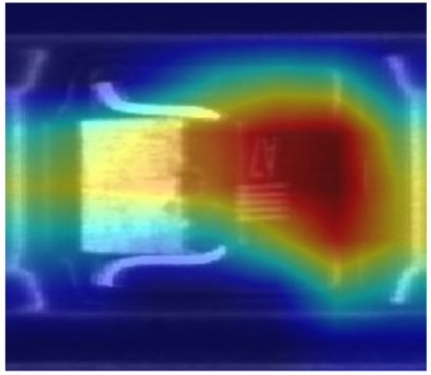


Figure 11: After backpropagation using wrong labels

Thanks
Q&A