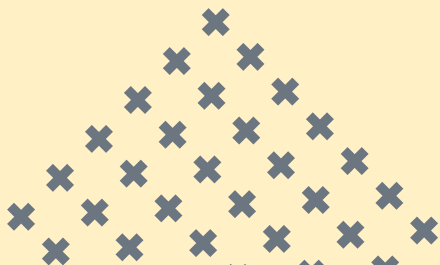




# Dynamic Inference Models

“Why waste time do a lot of compute,  
When few compute do trick”





# 01

## **WHAT/WHY DYNAMIC INFERENCE?**

What's dynamic inference and why would you even want to do it?



# 02

## **SKIP/DROP MODELS**

Models which choose to skip layers during inference

# 03

## **EARLY-EXIT MODELS**

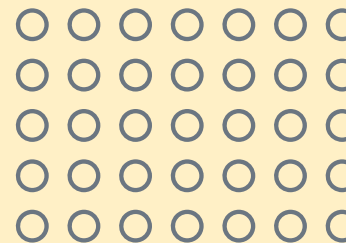
Models which cut-off computation using auxiliary classifiers

# 04

## **WHAT NEXT ?**

Possible research ideas





# 01

## DYNAMIC INFERENCE

“Become like water, my friend”

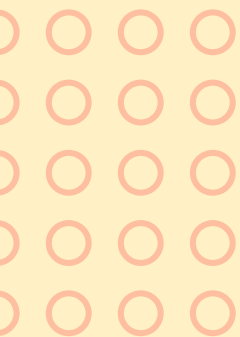


“A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.”



While the Lottery Ticket Hypotheses is restricted to pruning (not dynamic); we know that deep models are massively over-parameterized and have several redundant parameters. This over-parameterization is why deep learning in general and compression techniques like pruning have been quite successful.



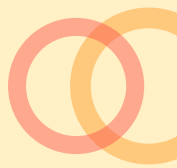


# Idea

Can it be possible to train a deep network and then to dynamically select useful sub-networks based on the input during test time so as to not perform redundant computations?

And that's what dynamic inference or conditional computation\* is about:

- Train a model such that during test time it is able to cleverly select parts of the network to execute depending on the input:
  - “Easy” input: Lower compute
  - “Hard” input: Higher compute



(a) Red wine

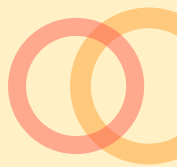


(b) Volcano

Easy ?





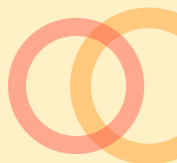


## Why Dynamic Inference?

- Resource-constrained environments e.g. mobiles
- Time-critical environments e.g. autonomous driving
- Unknown compute budget e.g. embedded systems

**Best case scenario:** Train a model without worrying about the end system and then at run time chose what parts to run or not depending on requirements.





## Dynamic Inference = How humans think\*

Refer to Eric's talk (from last year) on how we know that human mind does sequential processing and takes longer to process cognitively demanding problems

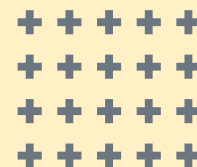
Humans = dynamic





# But first...

A (very) quick review of static methods of model compression:



## PRUNING

Cut-off redundant connections in a neural network



## KNOWLEDGE DISTILLATION

Train a smaller model from a larger model



## QUANTIZATION

Replace high precision floating point parameters of net with low precision



## DESIGN COMPACT MODELS (duh...)

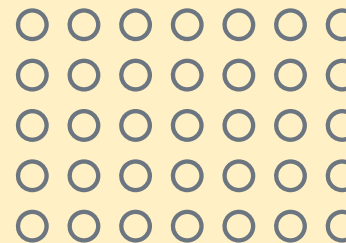
e.g. SqueezeNet, MobileNet





We shall see later how a lot of these ideas actually come in handy while doing dynamic inference

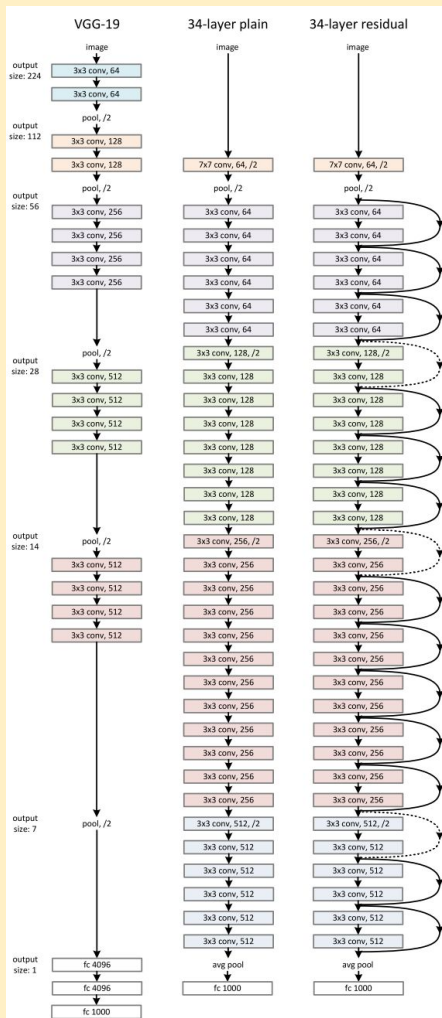




# 02

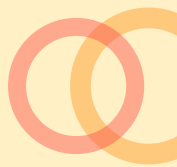
## SKIP/DROP MODELS

“All ~~models~~ residuals are wrong,  
But some are useful”



## ResNet refresher:

- Residual connections: Connections which skip layers, prevent vanishing gradient
- First to enable learning of very deep networks with 100+ layers (yay)



Residual connections = Over-parameterization:

- Residual connections enable the option of identity mappings which leads the network to have a lot of redundancy (parallel layers could be useless)
- Useful for training - prevent overfitting, avoid vanishing gradient - train very deep nets (\*1001 layers)
- Not useful during inference: can actually skip the layers parallel to the identity mappings without affecting results

**Residual much?**

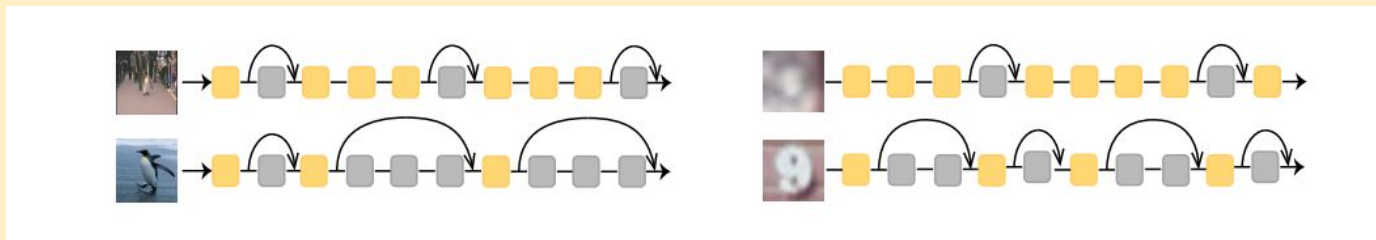
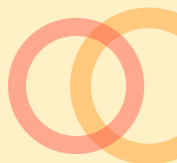


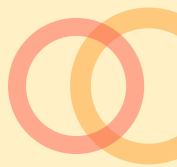




It would make sense if we were able to skip layers\*  
which are redundant and use the identity mappings.



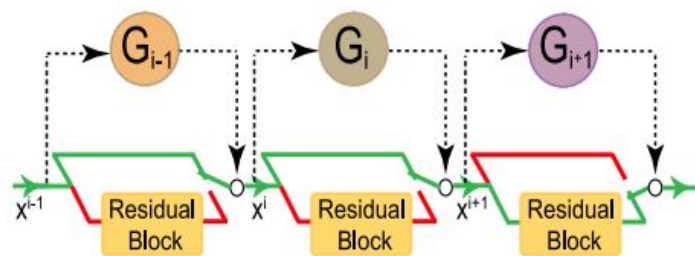
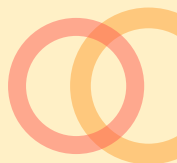




SkipNet idea:

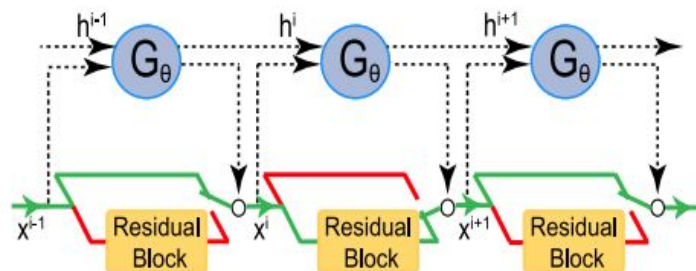
- Modify the layer connections to be gated (so as to switch on and off)
- Learn a policy to dynamically turn these switches on or off\*





(a) Feed-forward Gate

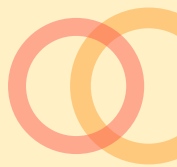
Each Res block has its own gate



(b) Recurrent Gate

A recurrent gate is shared across blocks





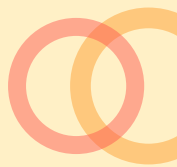
Skipping decisions:

- Choosing to skip/not skip is a discrete decision and non-differentiable\*
- Non-differentiable sequential decision making => Reinforcement learning
- Skipping policy: Given input and gate decide whether to skip or not

**Skipping policy**



\*(An option would be to use differentiable softmax approximation and switch to hard gates during test: sounds good, doesn't work)

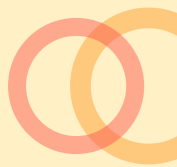


$$\begin{aligned}\min \mathcal{J}(\theta) &= \min \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{g}} L_{\theta}(\mathbf{g}, \mathbf{x}) \\ &= \min \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{g}} \left[ \mathcal{L}(\hat{y}(\mathbf{x}, F_{\theta}, \mathbf{g}), y) - \frac{\alpha}{N} \sum_{i=1}^N R_i \right]\end{aligned}$$

1st term: supervised loss

2nd term: policy gradient loss (R = reward of each gating module)

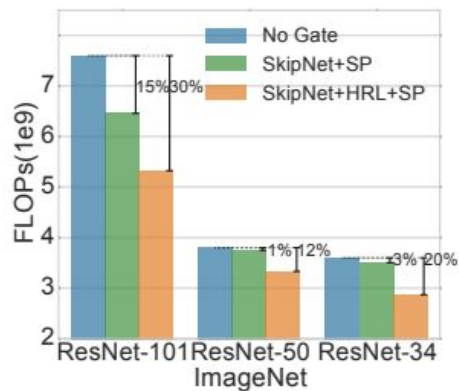
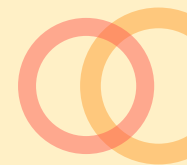




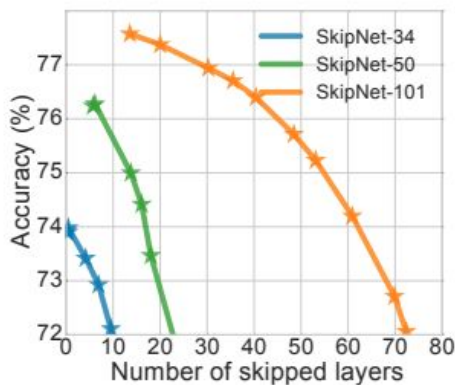
## Supervised pre-training:

- Hybrid loss actually gives worst of both worlds if used from scratch: neither good enough features (supervised) nor good enough policy (reinforcement) due to conflicting interests
- Pre-train: Relax gating to softmax in backward pass. Learn a good set of initial parameters

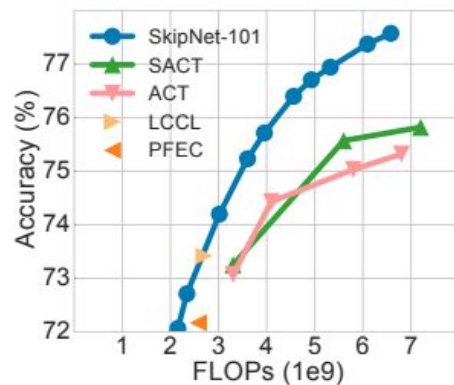




(a) Computation Reduction



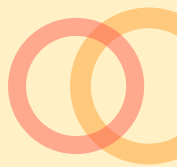
(b) Acc.-compt. Trade-off



(c) Comparison with Others



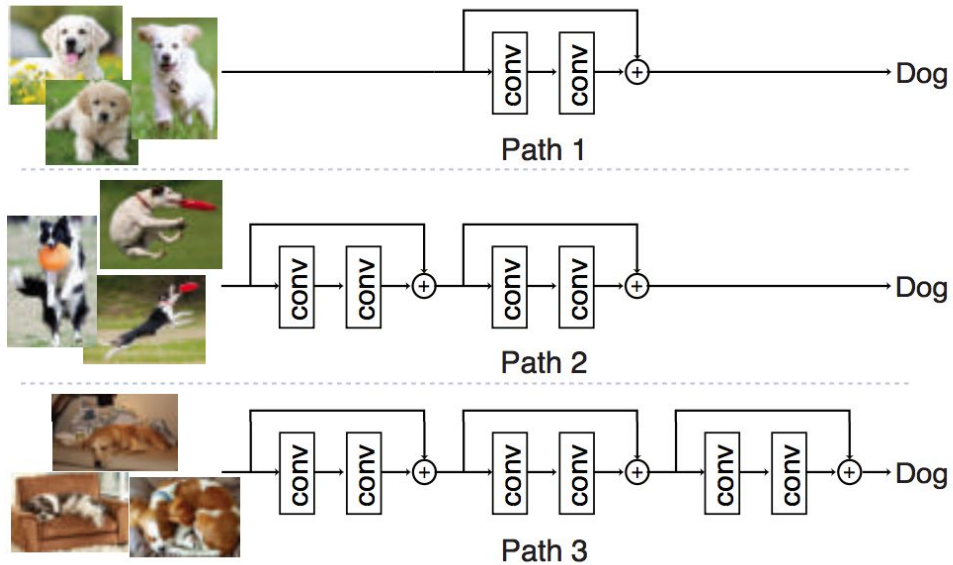
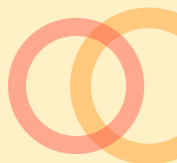


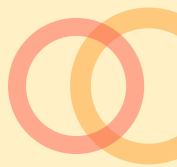


SkipNet issues:

- Additional parameters of gating mechanism (particularly for FF gate)
- Non-differentiable skipping decisions
- Two-stage training < End-to-end



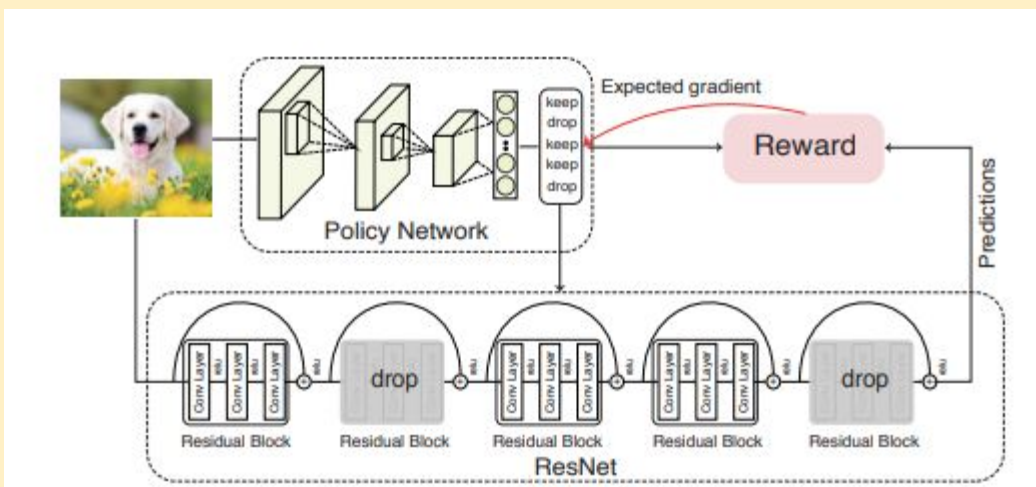
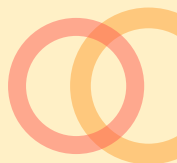




BlockDrop idea:

- Given pre-trained network -> Train policy to utilize minimum res blocks while maintaining accuracy





## BlockDrop Formulation



$$\pi_{\mathbf{W}}(\mathbf{u}|\mathbf{x}) = \prod_{k=1}^K \mathbf{s}_k^{\mathbf{u}_k} (1 - \mathbf{s}_k)^{1-\mathbf{u}_k}$$

$$\mathbf{s} = f_{pn}(\mathbf{x}; \mathbf{W}),$$

F = policy network, weights W

X = Image

s = network output after logit

s\_k = likelihood of kth residual block being dropped

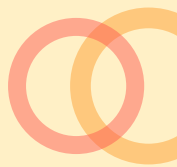
$$R(\mathbf{u}) = \begin{cases} 1 - (\frac{|\mathbf{u}|_0}{K})^2 & \text{if correct} \\ -\gamma & \text{otherwise.} \end{cases}$$

u\_k = whether kth block selected or dropped (action)

R = reward function

$$J = \mathbb{E}_{\mathbf{u} \sim \pi_{\mathbf{W}}} [R(\mathbf{u})].$$

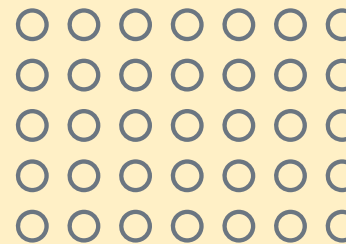
J = policy gradient loss



Two stage training:

- Curriculum learning: During epoch  $t$ , keep first  $K-t$  blocks on for policy gradients. This pre-trains the policy network for utility of each block
- Jointly finetune ResNet and Policy networks



A decorative graphic on the left side of the slide featuring three large, overlapping circles. The top circle is light blue, the middle circle is light yellow, and the bottom circle is light orange. They overlap in a vertical stack.

# 03

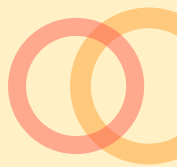
## EARLY EXIT MODELS

“Quit while you’re ahead”

**Idea:**

- Insert auxiliary classifiers at intermediate layers\*
- Return the output from auxiliary classifier as soon as the computation criteria is met (#operations, confidence threshold)

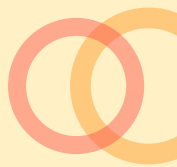




## Quick aside: Confidence

When we say confidence in deep learning, we use the probability of the class obtained from the softmax as proxy. So once the intermediate classifier outputs a probability 0.7 for class A we say that it's 70% confident that the input belongs to class A.





## Confidence vs Overconfidence

As shown by Guo et al, deep nets are poorly calibrated i.e. their confidence measures are often way off towards overconfidence.

### Why?

(Probably) overfitting the negative log-likelihood instead of the 0/1 classification loss.

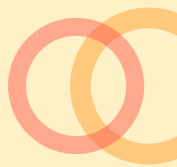
“Temperature scaling”: Post-processing method of scaling the output of the logits using a learned scaling factor (over validation)



## Idea:

- Insert auxiliary classifiers at intermediate layers
- Return the output from auxiliary classifier as soon as the computation criteria is met (#operations, confidence threshold)

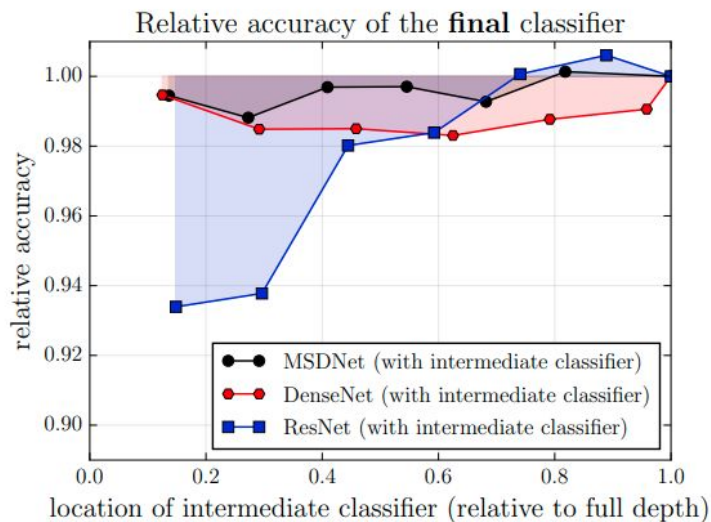
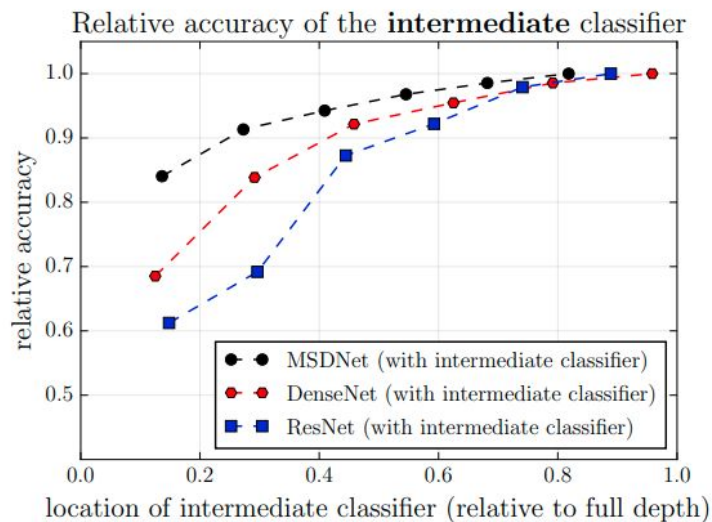
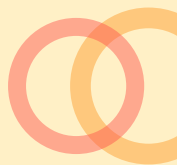




## Issues:

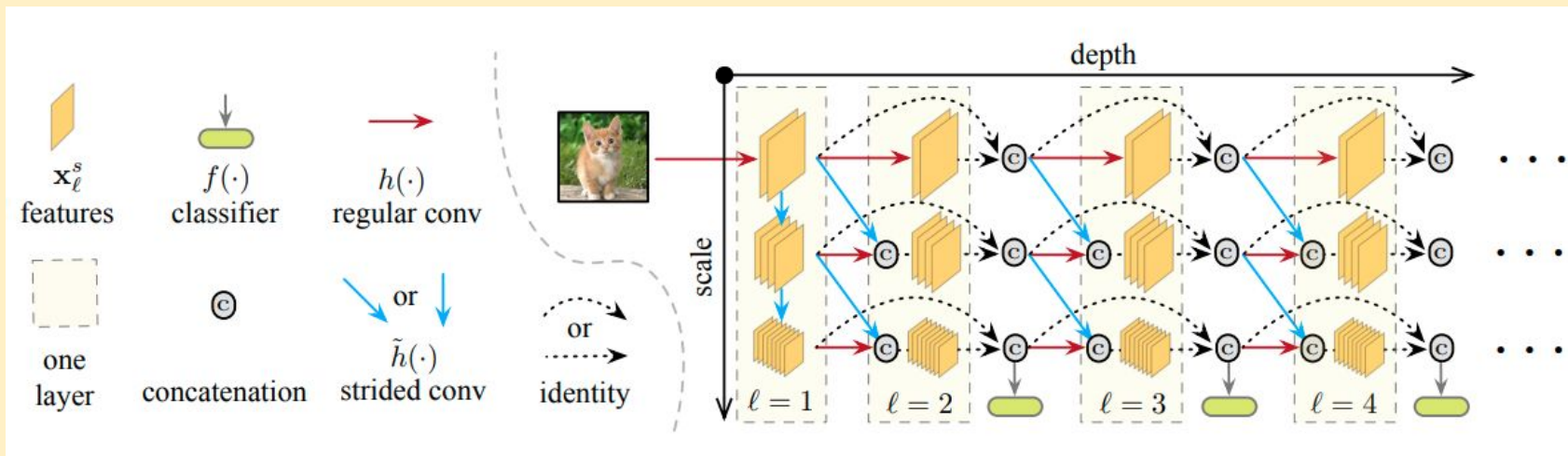
- Insert auxiliary classifiers at intermediate layers
  - Intermediate classifiers lead to poor performance as the network learns to optimize for better initial results
- Return the output from auxiliary classifier as soon as the computation criteria is met (#operations, confidence threshold)
  - Early classifiers have access to features learned in initial layers which are usually low-level and don't give a 'big-picture' view (quite literally)





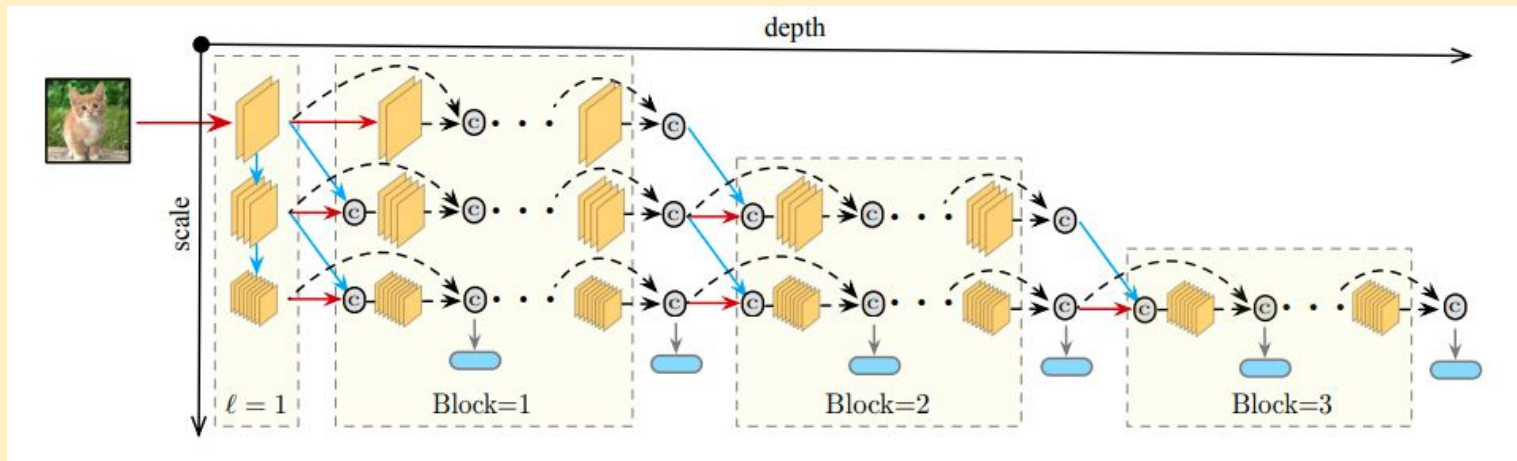
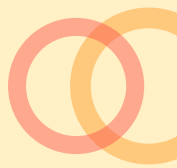
## Solutions:

- Intermediate classifiers lead to poor performance as the network learns to optimize for better initial results
  - Use Dense connections\* to connect each layer with all subsequent layers - allows final layers to bypass initial layers optimized for good initial results.
- Early classifiers have access to features learned in initial layers which are usually low-level
  - Learn multi-scale feature maps at each layer so as to have access to both coarse and fine features - best of both worlds

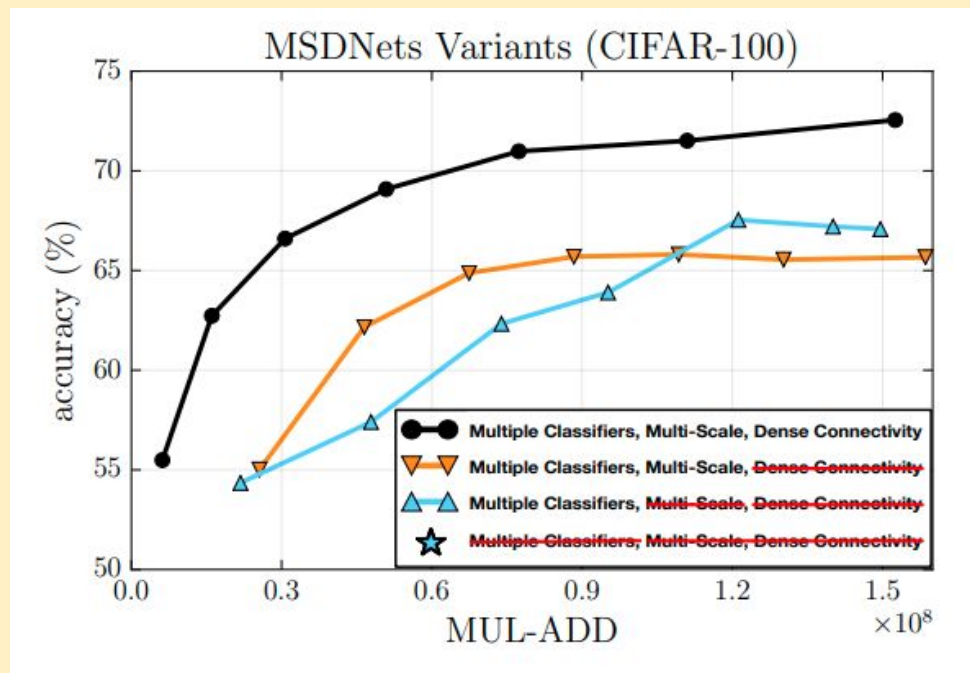


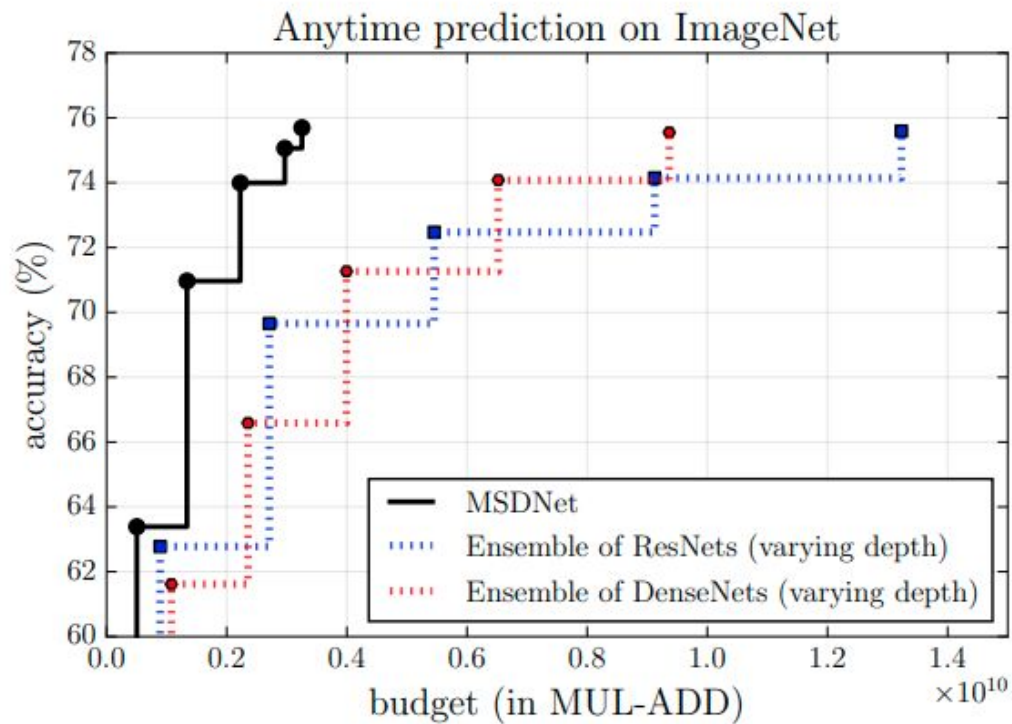
## MSDNet Architecture











**Issue:**

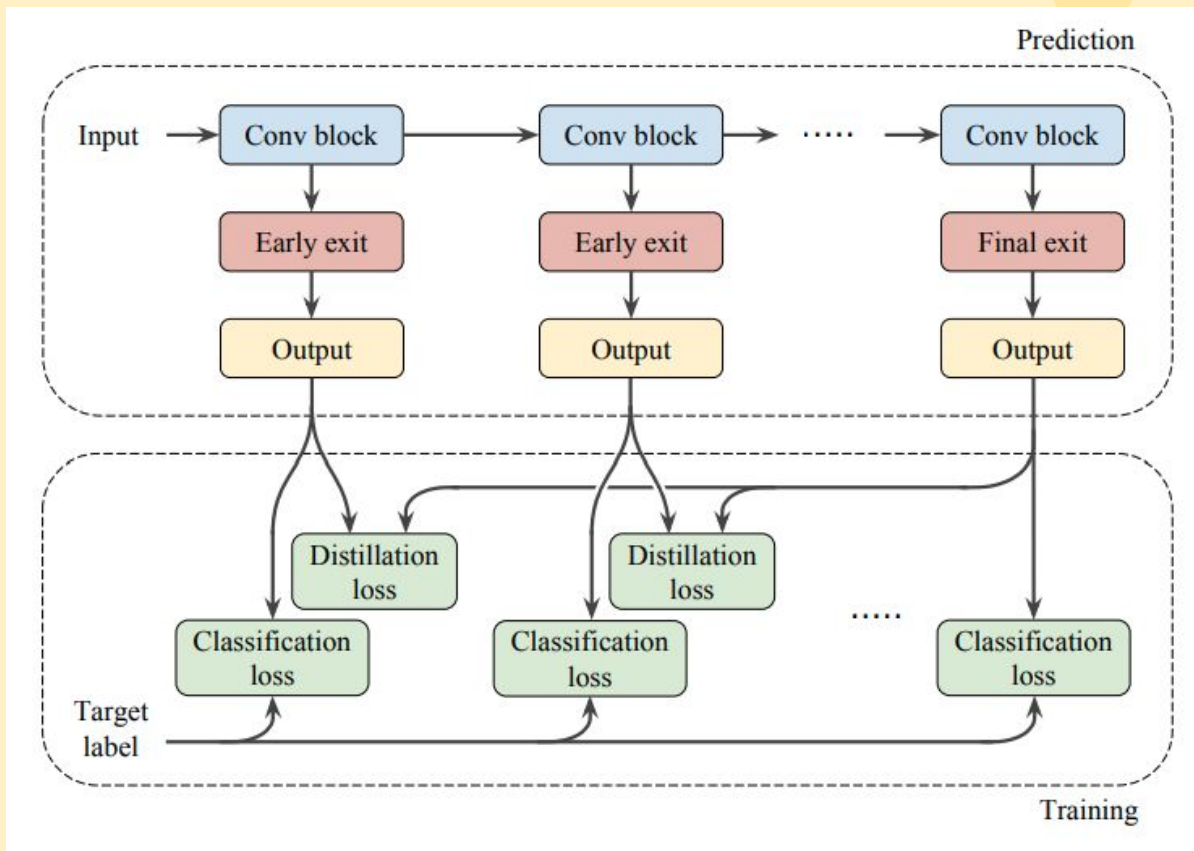
- Despite auxiliary classifiers at intermediate layers the final classifier still performs best

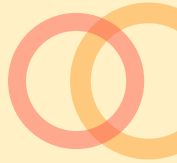


**Solution:**

- Despite auxiliary classifiers at intermediate layers the final classifier still performs best
  - Use self-distillation from later classifiers to train initial classifiers





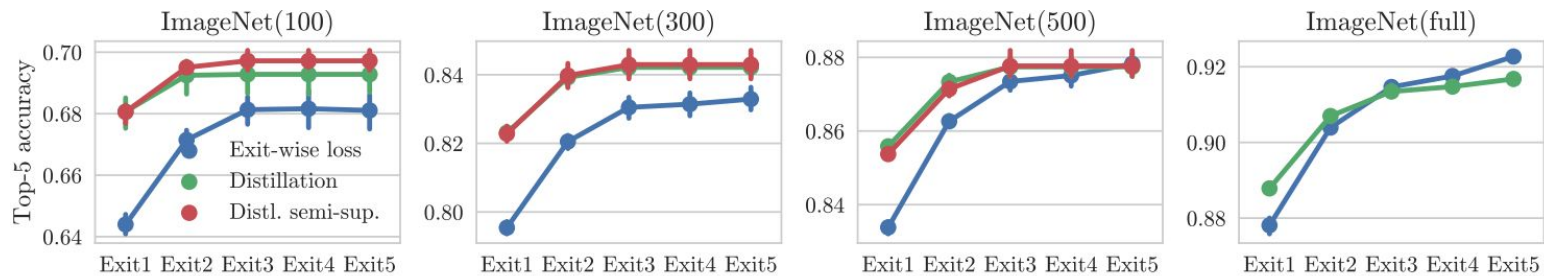
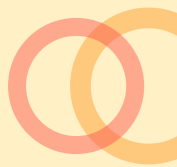


$$\mathcal{L}_{\text{dist}}(\mathbf{x}_n) = \frac{1}{M} \sum_{m=1}^M \frac{1}{|\mathcal{T}(m)|} \sum_{t \in \mathcal{T}(m)} \ell^{\tau}(\mathbf{p}_t(\mathbf{x}_n), \mathbf{p}_m(\mathbf{x}_n)),$$

$\mathbf{p}_t$  = teacher logits

$\mathbf{p}_m$  = student logits

$M$  = #classifiers

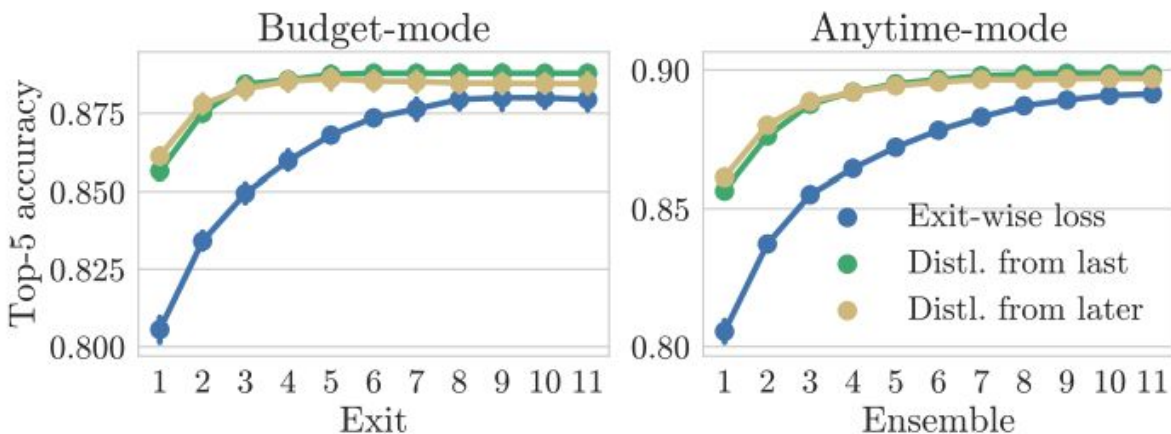


Distillation > Exit-wise loss



## Observations:

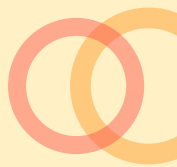
- Distillation from the last classifier is almost as good as distillation from all later classifiers
  - Makes sense. Last classifier has access to all high-level features





## Observations:

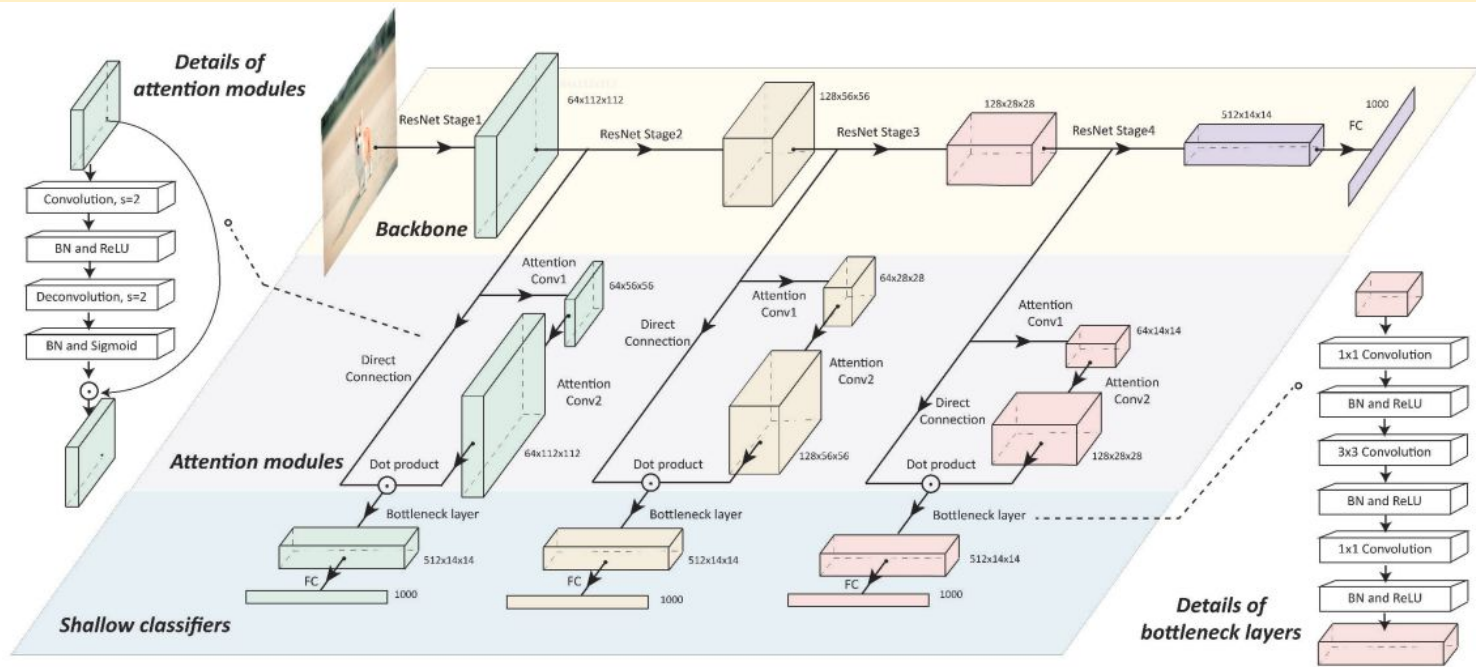
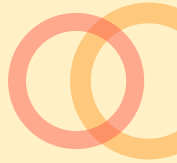
- Semi-supervised loss (only using distillation loss where labels are not available) does not add much to the learning of the model
  - Authors unsure about this

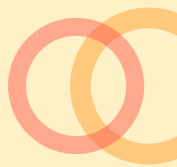


## Footnote:

- Exact same idea of using self-distillation also appears in the paper “Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation” at ICCV 2019.
- Furthermore, the authors of this paper add an additional attention module to develop their SCAN framework (discussed next).





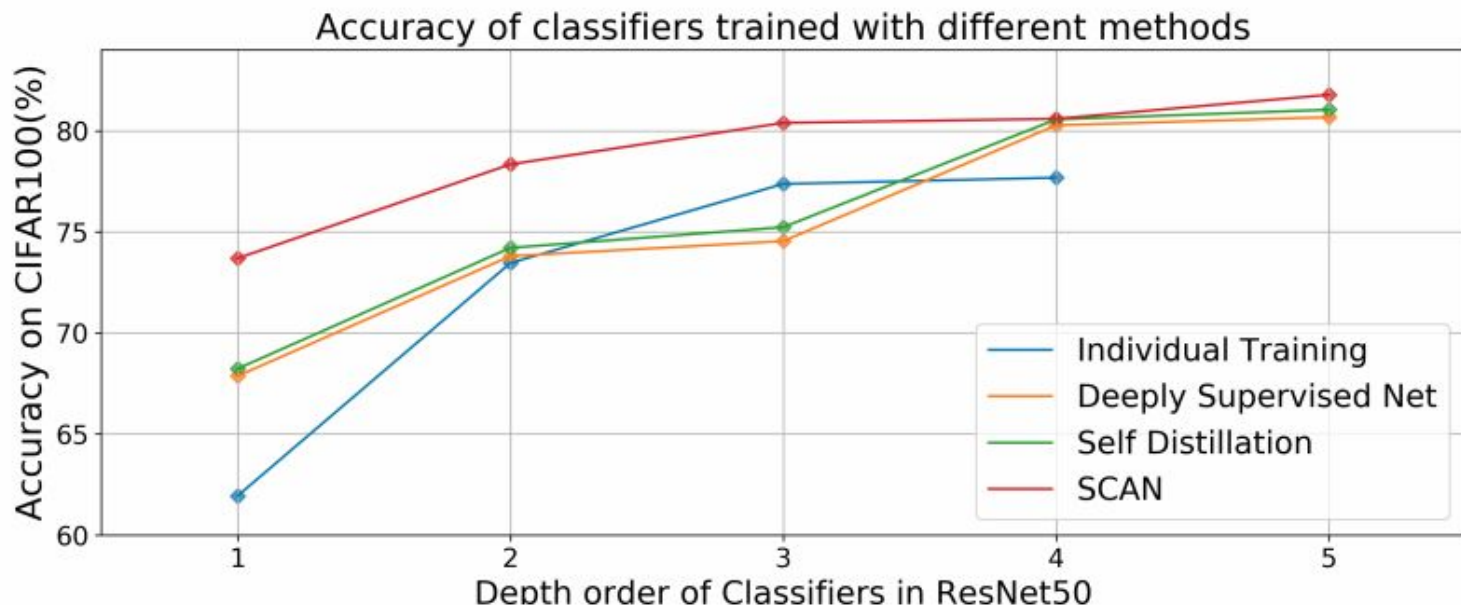
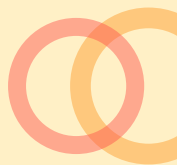


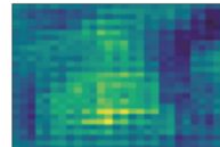
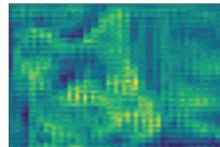
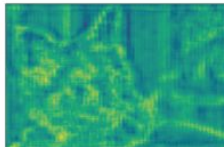
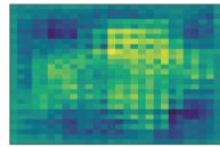
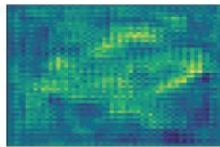
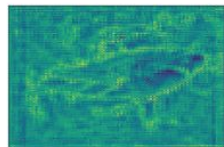
## SCAN Formulation:

$$\text{loss} = \sum_{i=1}^C \text{loss}_i = \sum_{i=1}^C \left( (1 - \alpha) \cdot \text{Cross Entropy}(q^i, y) + \alpha \cdot KL(q^i, q^C) + \lambda \cdot \|F_i - F_C\|_2^2 \right)$$

**Threshold calculation for each intermediate classifier:** Using genetic programming







Input image

Attention 1

Attention 2

Attention 3

Initial classifiers = outlines i.e. local & high frequency info


Later classifiers = texture i.e. global and low frequency info

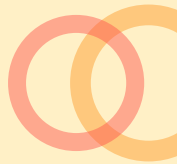


# 04

## WHAT NEXT?

The wild wild west  
(of my research ideas)

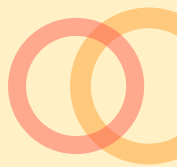




## Idea 1: Given a trained static network convert it into a dynamic network

- Why?
  - Because...
  - Need to come up with use cases where training data might be unavailable but trained model is.
- How?
  - Zero-shot Knowledge Distillation\*
  - Idea: CNNs learn impressions of class labels during training which can be distilled without access to any training data
- Issues?
  - Motivation

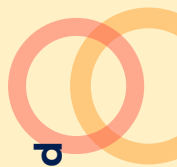




## Idea 2: Meta learning of conditional computation models

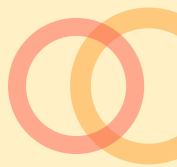
- Why ?
  - Training data efficiency  $\subseteq$  Resource efficiency
  - In niche applications where these models will actually be deployed few-shot learning would be needed
- How?
  - In the meta-learning formulation use each auxiliary classifier as a task
  - The meta-task would be learn parameters which quickly generalize to each early exit
- Issues?
  - Usually the entire parameter space is shared between the different tasks in meta-learning: not the case here





### Idea 3: Few-shot/Zero-shot learning using self-supervision

- Why ?
  - Training data efficiency  $\subseteq$  Resource efficiency
  - In niche applications where these models will actually be deployed few-shot learning would be needed
- How?
  - Use self-supervision
  - Explicitly force the representations in between initial layers & auxiliary classifiers to be similar in early exits and final exit (not the backbone network)
- Issues?
  - How to change the final classifier such that the benefits of self-supervision can actually be utilized (since self-supervision changes the classifier)



## Idea 4: Pruning for dynamic inference

- Why?
  - Later layers have better coarse features -> build early exit models with connections to later layers and then prune them before test
- How?
  - Instead of self-attention in early layers use attention on later layers
  - Since this attention is not available during inference, distill it into the parallel auxiliary classifier ?
- Issues?
  - Not sure about how really
  - Will parallel distillation from attention work?
  - Any inputs regarding 'detachable' attention modules?

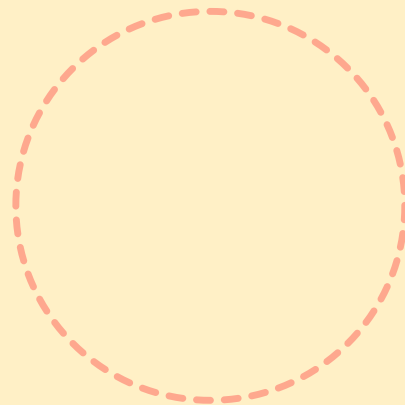




# THANKS!

Questions?

(References and slides will be  
made available on MLRG slack;  
also plan to write a blog post)



CREDITS: This presentation template was created  
by Slidesgo, including icons by Flaticon, and  
infographics & images by Freepik.

