

# ELimNet: Elimination based Lightweight Neural Net with Pretrained Weights

#Computer\_Vision

#Lightweight

#Finetuning

HappyFace

HyunSoo Kim, JaeYoung Jeon, JoonHong Kim,  
SeongWook Choi, YeonJoo Kim, YoungJin Ahn

## ELimNet: Elimination based Lightweight Neural Network with Pretrained Weights

Why

- Pretraining empty models constructed with AutoML NAS with Optuna costs too much time.
- Empty model underperforms pretrained models in terms of classification performance.
- Pretrained models underperforms empty NAS models in terms of inference time and in #params.

What

- Adapting optimizing method for pretrained transformers models: dropping top layers.
- Removing Convolutional Top Layers from Pretrained Models(EfficientNet B0, ResNet18)
- Lightweight models with less than 1 Million #params with pretrained weights are proven to be effective.

How

- Fetch pretrained torchvision models.
- Remove convolutional layers that are close to the loss function.
- Resize fully connected layers according to the remain top convolutional layer.
- Compare performance between original pretrained models, NAS models based on loss, accuracy and f1 score.
- Measure inference CPU and CUDA time.

# Table of Contents

- 01 Costs of Time
- 02 ELimNet Method
- 03 Performance



# Costs of Time

---

# Costs of Time

Backbone, Hyperparameters

Configuration  
 $\lambda$



Black box

Train with  $\lambda$



Accuracy, Speed, Size, ....

Evaluate Objective  
 $f(\lambda)$



Update  
Acquisition Function



Update  
Surrogate Function



Bayesian Optimization is effective to search lightweight models, but costs too much time.

# Costs of Time

Input	Operator	<i>t</i>	<i>c</i>	<i>n</i>	<i>s</i>
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Mobilenetv2



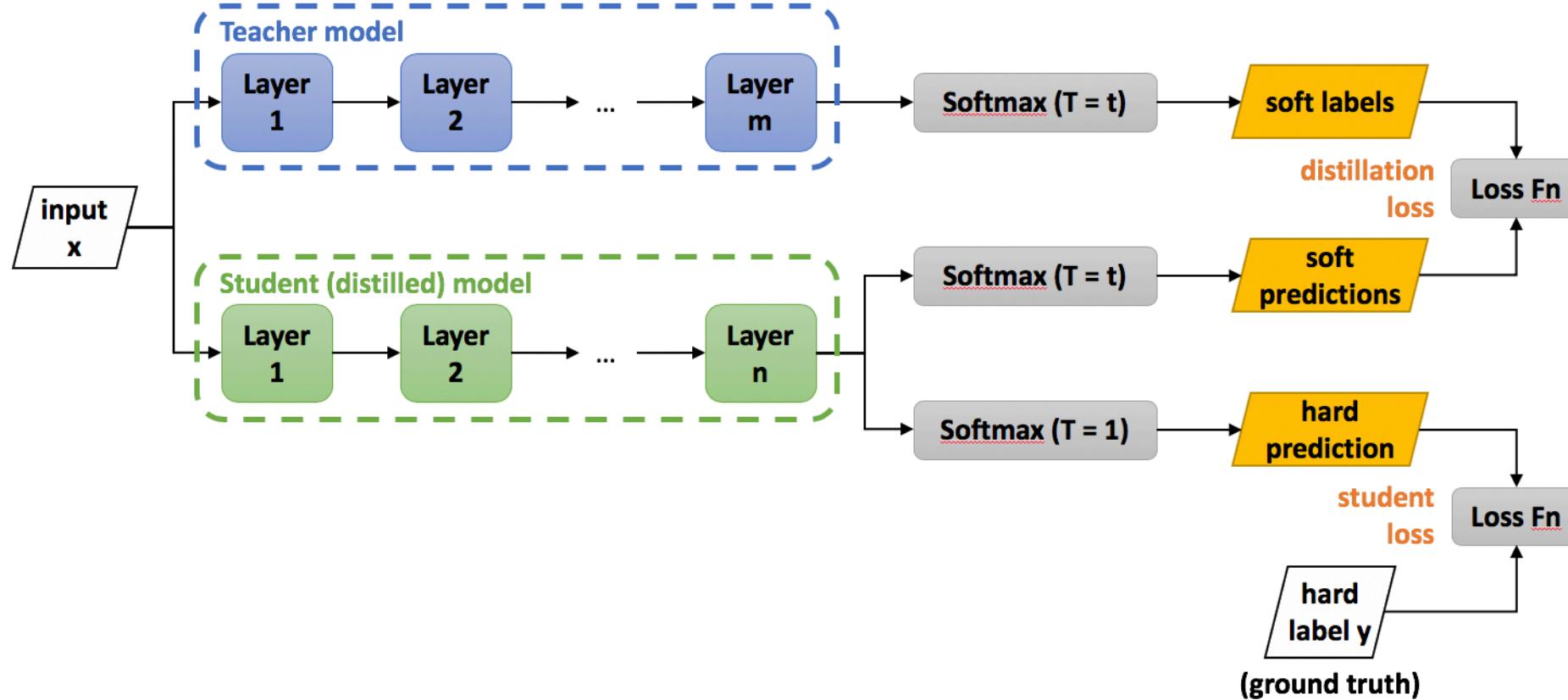
## BACKBONE:

```
[  
    # [repeat, module, args]  
    [1, Conv, [32, 3, 2]],  
    # InvertedResidualv2: [c, t, n, s]  
    [1, InvertedResidualv2, [16, 1, 1]],  
    [2, InvertedResidualv2, [24, 6, 2]],  
    [3, InvertedResidualv2, [32, 6, 2]],  
    [4, InvertedResidualv2, [64, 6, 2]],  
    [3, InvertedResidualv2, [96, 6, 1]],  
    [3, InvertedResidualv2, [160, 6, 2]],  
    [1, InvertedResidualv2, [320, 6, 1]],  
    [1, Conv, [1280, 1, 1]],  
    [1, GlobalAvgPool, []],  
    [1, Conv, [1000, 1, 1, null, 1, null]]  
]
```

mobilenetv2.yaml

Even if the model is built, it is empty model without pretrained weights.

# Costs of Time

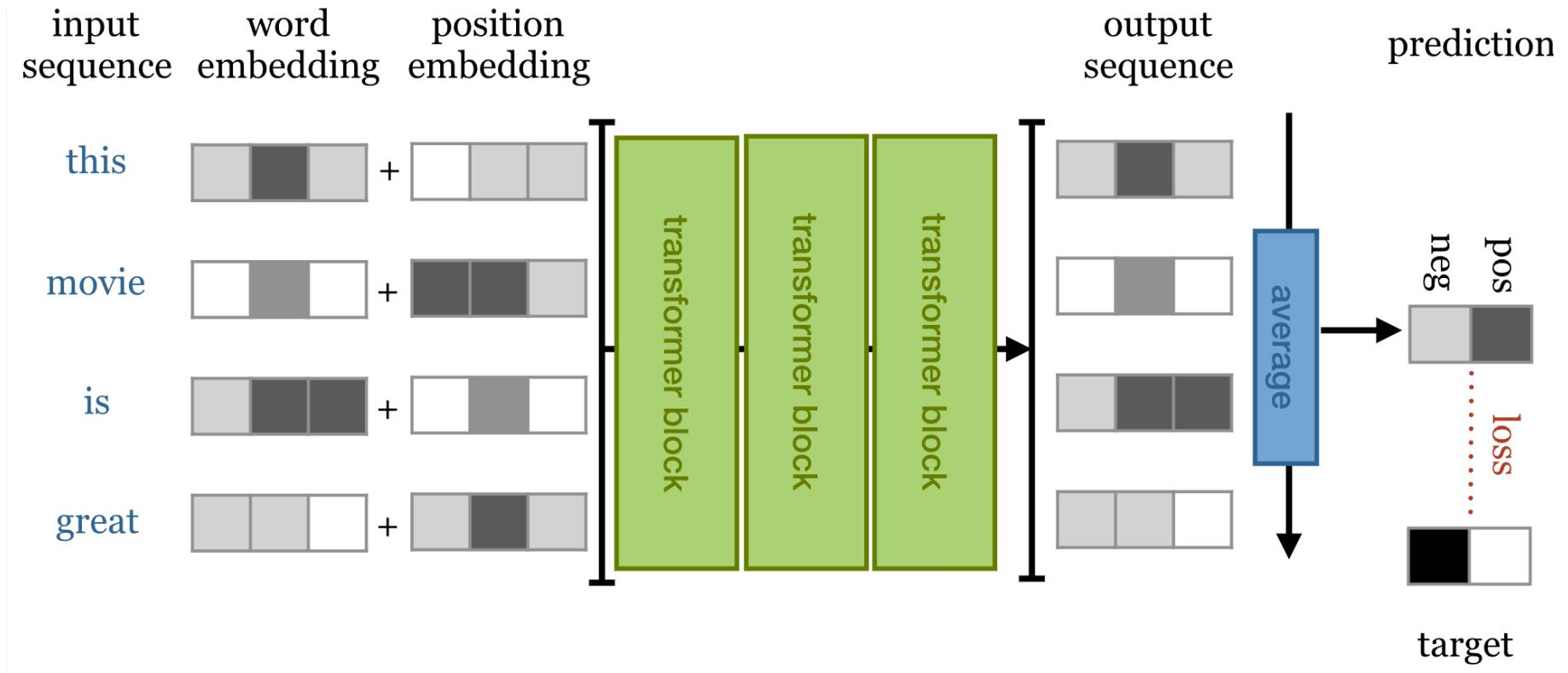


Knowledge distillation from pretrained teacher model boosts performance but still requires 200+ epochs

# ELimNet Method

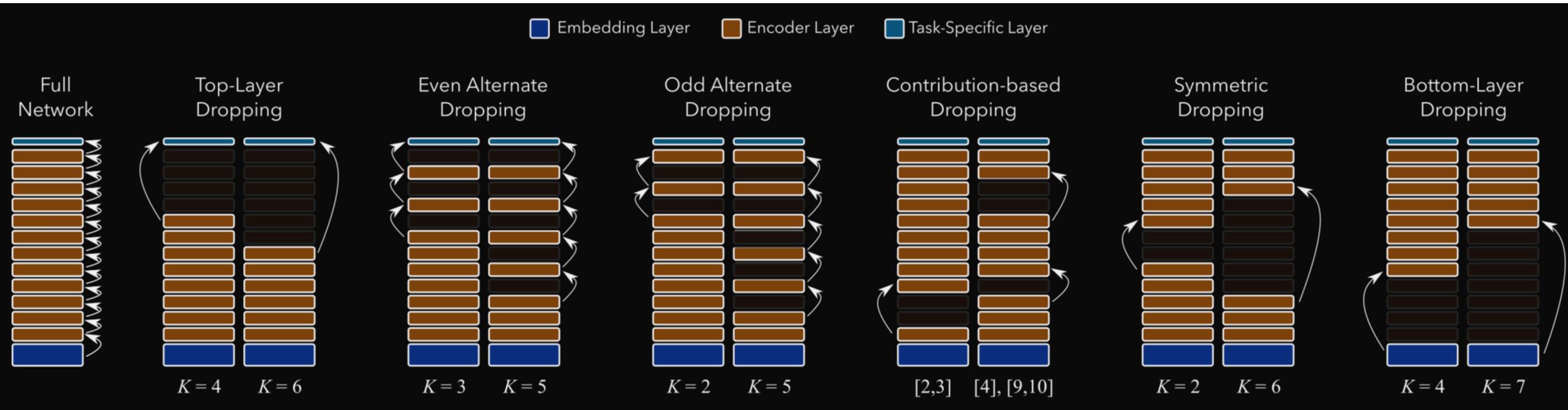
---

# ELimNet Method



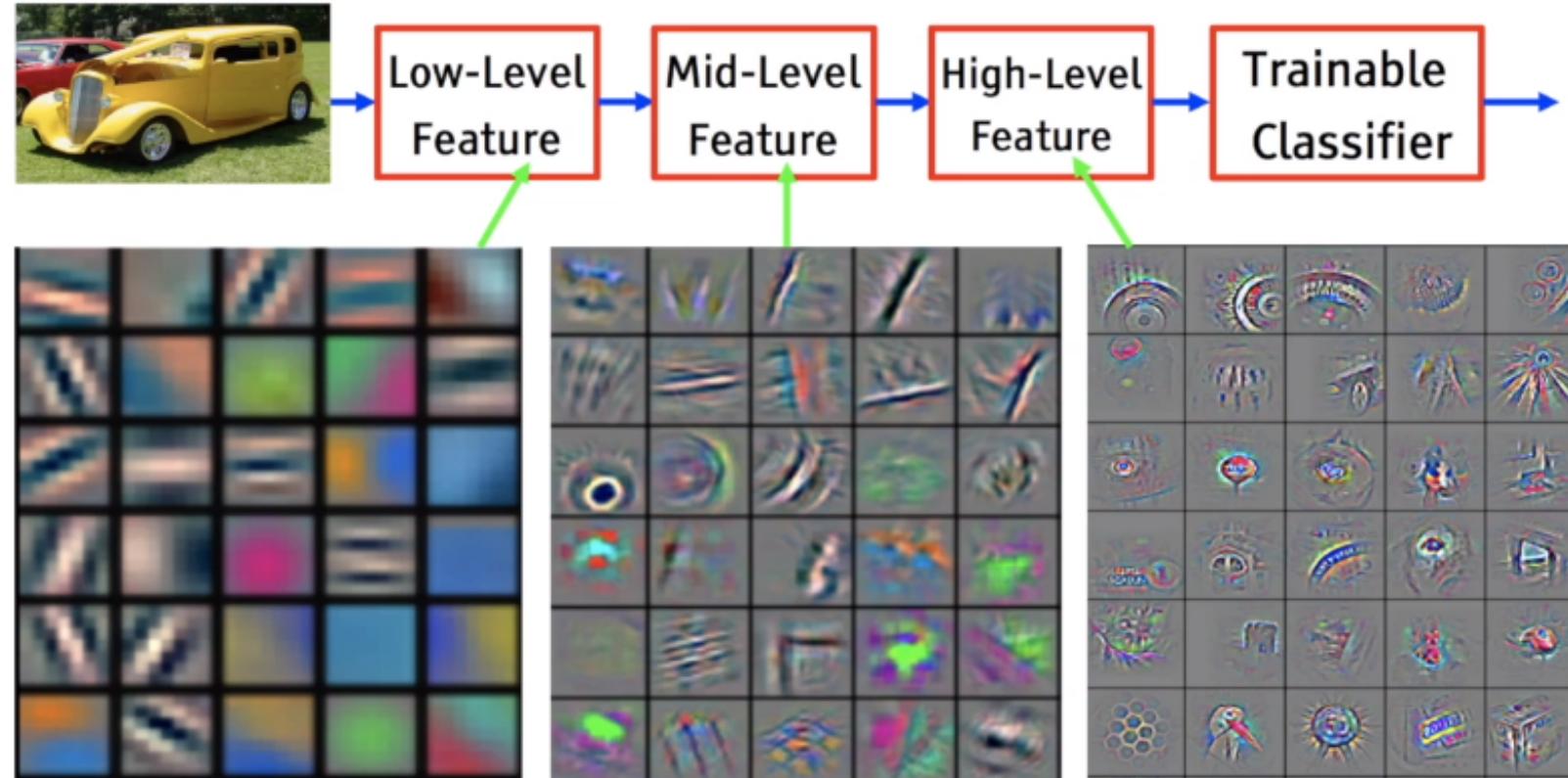
Transformers layers that are close to embeddings are embedding layers, and layers that are close to the loss is task-specific layers.

# ELimNet Method



Removing pretrained transformers top layers may yield 40% reduction in size,  
while preserving up to 98.2% of the performance.

# ELimNet Method



Lower layers represent simple aspects of the image, Higher layers can represent sophisticated aspects. Therefore, eliminated higher layers and used lower layers with pretrained weights for downstream task.

# ELimNet Method

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	$7 \times 7, 64$ , stride 2
		$3 \times 3$ max pool, stride 2
conv2_x	$56 \times 56 \times 64$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	$7 \times 7$ average pool
fully connected	1000	$512 \times 1000$ fully connections
softmax	1000	

Stage $i$	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$
1	Conv3x3	$224 \times 224$
2	MBConv1, k3x3	$112 \times 112$
3	MBConv6, k3x3	$112 \times 112$
4	MBConv6, k5x5	$56 \times 56$
5	MBConv6, k3x3	$28 \times 28$
6	MBConv6, k5x5	$14 \times 14$
7	MBConv6, k5x5	$14 \times 14$
8	MBConv6, k3x3	$7 \times 7$
9	Conv1x1 & Pooling & FC	$7 \times 7$

Selected ResNet18 and EfficientNet B0 as pretrained models.

# ELimNet Method

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	$7 \times 7, 64$ , stride 2
		$3 \times 3$ max pool, stride 2
conv2_x	$56 \times 56 \times 64$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	$7 \times 7$ average pool
fully connected	1000	$512 \times 1000$ fully connections
softmax	1000	

Stage <i>i</i>	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$
1	Conv3x3	$224 \times 224$
2	MBConv1, k3x3	$112 \times 112$
3	MBConv6, k3x3	$112 \times 112$
4	MBConv6, k5x5	$56 \times 56$
5	MBConv6, k3x3	$28 \times 28$
9	Conv1x1 & Pooling & FC	$7 \times 7$

Removing convolutional top layers from pretrained models for image classification task.

# ELimNet Method

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	$7 \times 7, 64$ , stride 2
conv2_x	$56 \times 56 \times 64$	$3 \times 3$ max pool, stride 2 $\left[ \begin{array}{c} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 2$
conv3_x	$28 \times 28 \times 128$	$\left[ \begin{array}{c} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 2$
average pool	$1 \times 1 \times 512$	$7 \times 7$ average pool
fully connected	1000	$512 \times 1000$ fully connections
softmax	1000	

```

● ● ●

class ElimResNet18(nn.Module):
    def __init__(self):
        super(ElimResNet18, self).__init__()
        self.model = models.resnet18(pretrained=True)
        del self.model.layer3
        del self.model.layer4

        self.num_in_features = 128 # replace fully connected layers
        self.model.fc = nn.Linear(self.num_in_features, 6)

    def _forward_impl(self, x):
        x = self.model.conv1(x)
        x = self.model.bn1(x)
        x = self.model.relu(x)
        x = self.model.maxpool(x)

        x = self.model.layer1(x)
        x = self.model.layer2(x)
        # x = self.model.layer3(x)
        # x = self.model.layer4(x)

        x = self.model.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.model.fc(x)
        return x

    def forward(self, x):
        return self._forward_impl(x)

```

Resizing fully connected layers accordingly

# Performance of ELimNet

---

## Pretrained Models vs ELimNet (Inference)

	# of Parameters	# of Layers	CPU times (sec)	CUDA time (sec)	Submitted Time
Pretrained EfficientNet B0	4.0M	352	3.9s	4.0s	105.6819
EfficientNet B0 Elim 2	0.9M	245	4.1s	13.0s	83.4161
EfficientNet B0 Elim 3	0.30M	181	<b>3.0s</b>	9.0s	<b>73.4604</b>
Resnet18	11.17M	69	-	-	-
Resnet18 Elim 2	0.68M	37	-	-	-

Construct lightweight CNN model with less than 1M #params, saves inference time by 20%.

# Pretrained Models vs ELimNet (Train)

ch

k

[100 epochs]	# of Parameters	# of Layers	Train	Validation	Submitted F1
Pretrained EfficientNet B0	4.0M	352	Loss: 0.43 Acc: 81.23 F1: 0.77	Loss: 0.469 Acc: 82.17 F1: 0.76	0.7493
EfficientNet B0 Elim 2	0.9M	245	loss:0.652, acc: 87.22%, <b>f1: 0.84</b>	Loss: 0.622, Acc: 80.14% <b>F1: 0.77</b>	<b>0.7603</b>
EfficientNet B0 Elim 3	<b>0.30M</b>	<b>181</b>	Loss: 0.602, Acc: 78.17% F1: 0.74	Loss: 0.661, Acc: 77.41% F1: 0.74	0.7349
Resnet18	11.17M	69	Loss: 0.578, Acc: 78.90% <b>F1: 0.76</b>	Loss: 0.700, Acc: 76.17% F1: 0.719	-
Resnet18 Elim 2	<b>0.68M</b>	<b>37</b>	<b>Loss: 0.447,</b> <b>Acc: 83.73%</b> F1: 0.71	Loss: 0.712, Acc: 75.42% F1: 0.71	-

# Pretrained Models vs ELimNet (Train)

ch

k

[100 epochs]	# of Parameters	# of Layers	Train	Validation	Submitted F1
Pretrained EfficientNet B0	4.0M	352	Loss: 0.43 Acc: 81.23 F1: 0.77	Loss: 0.469 Acc: 82.17 F1: 0.76	0.7493
EfficientNet B0 Elim 2	0.9M	245	loss:0.652, acc: 87.22%, f1: 0.74	Loss: 0.622, Acc: 80.14% F1: 0.77	0.7603
EfficientNet B0 Elim 2	0.20M	181	Loss: 0.602 Acc: 78.17% F1: 0.74	Loss: 0.661 Acc: 77.41% F1: 0.74	0.7349
Resnet18	11.17M	69	Loss: 0.578, Acc: 78.90% F1: 0.76	Loss: 0.700, Acc: 76.17% F1: 0.719	-
Resnet18 Elim 2	0.68M	37	Loss: 0.447, Acc: 83.73% F1: 0.71	Loss: 0.712, Acc: 75.42% F1: 0.71	-

Reduces parameters to 7%(or less)

of its original parameters while maintaining its performance.

## NAS + HPO vs ELimNet (Inference)

	# of Parameters	# of Layers	CPU times (sec)	CUDA time (sec)	Submitted time
Empty MobileNet V3	4.2M	227	4	13	-
Empty EfficientNet B0	1.3M	352	3.780	3.782	68.3922
Empty DWConv & InvertedResidualv3 NAS	0.08M	66	1	3.5	61.1138
Empty MBConv NAS	0.33M	141	2.14	7.201	67.0518
EfficientNet B0 Removed 3	0.30M	181	3.0s	9s	73.4604
Resnet18 Removed 2	0.68M	37	-	-	-

Maintaining low layers' pretrained weights underperformed in inference time to NAS & HPO model searched with Optuna.

# Empty NAS + HPO Models vs ELimNet (Train)

[100 epochs]	# of Parameters	# of Layers	Train	Valid	Submitted F1
Empty MobileNet V3	4.2M	227	Loss 0.925 Acc: 65.18 F1: 0.58	Loss 0.993 Acc: 62.83 F1: 0.56	-
Empty EfficientNet B0	1.3M	352	Loss 0.867 Acc: 67.28 F1: 0.61	Loss 0.898 Acc: 66.80 F1: 0.61	0.6337
Empty DWConv & InvertedResidualv3 NAS	0.08M	66	-	Loss: 0.766 Acc: 71.71 F1: 0.68	0.6740
Empty MBConv NAS	0.33M	141	Loss: 0.786 Acc: 70.72 F1: 0.66	Loss: 0.866 Acc: 68.09 F1: 0.62	0.6245
Resnet18 Removed 2	0.68M	37	Loss: 0.447 Acc: 83.73% F1: 0.71	Loss: 0.712 Acc: 75.42% F1: 0.71	0.7603
EfficientNet B0 Removed 3	0.30M	181	Loss: 0.602 Acc: 78.17% F1: 0.74	Loss: 0.661 Acc: 77.41% F1: 0.74	0.7349

# Empty NAS + HPO Models vs ELimNet (Train)

[100 epochs]	# of Parameters	# of Layers	Train	Valid	Submitted F1
Empty MobileNet V3	4.2M	227	Loss: 0.925 Acc: 65.18 F1: 0.58	Loss: 0.993 Acc: 62.83 F1: 0.56	-
Empty EfficientNet B0	1.3M	352	Loss: 0.867 Acc: 67.28 F1: 0.61	Loss: 0.898 Acc: 66.80 F1: 0.61	0.6337
Empty DWConv & InvertedResidualv3	0.08M	66	-	Loss: 0.766 Acc: 71.55 F1: 0.68	0.6740
Empty MBConv NAS	0.33M	141	Loss: 0.786 Acc: 70.72 F1: 0.66	Loss: 0.866 Acc: 68.09 F1: 0.62	0.6245
Resnet18 Removed 2	0.68M	37	Loss: 0.447 Acc: 83.73% F1: 0.71	Loss: 0.712 Acc: 75.42% F1: 0.71	0.7603
EfficientNet B0 Removed 3	0.30M	181	Loss: 0.602 Acc: 78.17% F1: 0.74	Loss: 0.661 Acc: 77.41% F1: 0.74	0.7349

Filled can is better than empty can.

# Work in Progress

- Will add ResNet18's inference time data and compare with Optuna NAS constructed lightweight model.
- Will test on pretrained MobileNetV3, MnasNet on torchvision.
- Will be applied on other small datasets such as Fashion MNIST dataset and Plant Village dataset.

<https://github.com/snoop2head/elimnet>



T-shirt/top



T-shirt/top



Potato\_\_Early\_blight (20)



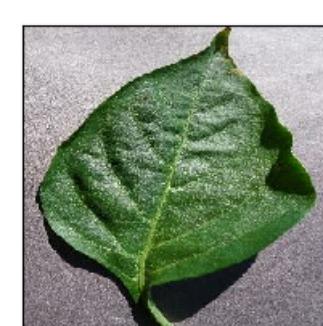
Apple\_\_healthy (3)



Sneaker



Pullover



Pepper,\_bell\_\_healthy (10)



Tomato\_\_Spider\_mites Two-spotted\_spider\_

## End Remark

Steve Rogers : Big man in a suit of armour. Take that off, what are you?  
Tony Stark : Genius, billionaire, playboy, philanthropist.



# Thank you

## ELimNet: Elimination based Lightweight Neural Network with Pretrained Weights

Why

- Pretraining empty models constructed with AutoML NAS with Optuna costs too much time.
- Empty model underperforms pretrained models in terms of classification performance.
- Pretrained models underperforms empty NAS models in terms of inference time and in #params.

What

- Adapting optimizing method for pretrained transformers models: dropping top layers.
- Removing Convolutional Top Layers from Pretrained Models(EfficientNet B0, ResNet18)
- Lightweight models with less than 1 Million #params with pretrained weights are proven to be effective.

How

- Fetch pretrained torchvision models.
- Remove convolutional layers that are close to the loss function.
- Resize fully connected layers according to the remain top convolutional layer.
- Compare performance between original pretrained models, NAS models based on loss, accuracy and f1 score.
- Measure inference CPU and CUDA time.

# Q&A